U N I V E R S I D A D A N D I N A
NÉSTOR CÁCERES VELÁSQUEZ
ESCUELA DE POSGRADO
MAESTRÍA EN INGENIERÍA DE SISTEMAS
MENCIÓN: INGENIERÍA DE SOFTWARE

T E S I S

ESTUDIO COMPARATIVO DE PLATAFORMAS MIDDLEWARE OPEN-SOURCE DE INTERNET DE LAS COSAS (IoT) BASADO EN MÉTRICAS CUANTITATIVAS Y CUALITATIVAS

PRESENTADA POR

ANTONIO ARROYO PAZ

PARA OPTAR EL GRADO ACADÉMICO DE

MAGÍSTER EN INGENIERÍA DE SISTEMAS

JULIACA – PERÚ
2018

U N I V E R S I D A D A N D I N A
NÉSTOR CÁCERES VELÁSQUEZ
ESCUELA DE POSGRADO
MAESTRÍA EN INGENIERÍA DE SISTEMAS
MENCIÓN: INGENIERÍA DE SOFTWARE

T E S I S

ESTUDIO COMPARATIVO DE PLATAFORMAS MIDDLEWARE OPEN-SOURCE DE INTERNET DE LAS COSAS (IoT) BASADO EN MÉTRICAS CUANTITATIVAS Y CUALITATIVAS.

PRESENTADA POR
ANTONIO ARROYO PAZ

PARA OPTAR EL GRADO ACADÉMICO DE
MAGÍSTER EN INGENIERÍA DE SISTEMAS

APROBADA POR EL JURADO:

PRESIDENTE : .....
M.Sc. JUAN CARLOS HERRERA MIRANDA

PRIMER MIEMBRO : .....
Mgtr. ALCIDES VELÁSQUEZ ARI

SEGUNDO MIEMBRO : .....
Mgtr. HENRY ÓSCAR VALENCIA GAMARRA

ASESOR DE TESIS : .....
DR. KATTY AGRIPINA PÉREZ ORDOÑEZ

UNIVERSIDAD ANDINA
"NÉSTOR CÁCERES VELÁSQUEZ"
ESCUELA DE POSGRADO

RESOLUCIÓN DIRECTORAL Nº 1040–2018-USA-EPG/UANCV

Juliaca, 2018 Octubre 16

VISTOS:

El expediente Nº 21457 del (a) Bachiller ARROYO PAZ ANTONIO, con número de DNI. 01213151 y con número de matrícula 23120005, de la MAESTRÍA en INGENIERÍA DE SISTEMAS, Mención: INGENIERÍA DE SOFTWARE, de la Escuela de Posgrado de la Universidad Andina "Néstor Cáceres Velásquez" de Juliaca;

CONSIDERANDO:

Que, el (a) Bach. ARROYO PAZ ANTONIO, con número de matrícula 23120005 de la Maestría en INGENIERÍA DE SISTEMAS, Mención: INGENIERÍA DE SOFTWARE, de la Escuela de Posgrado de la Universidad Andina Néstor Cáceres Velásquez de Juliaca; ha Solicitado la Sustentación del Dictamen de Tesis titulada: ESTUDIO COMPARATIVO DE PLATAFORMAS MIDDLEWARE OPEN-SOURCE DE INTERNET DE LAS COSAS (IoT) BASADO EN MÉTRICAS CUANTITATIVAS Y CUALITATIVAS. Para ser sustentada;

Que, el (a) referido (a) Dictamen de Tesis aprobado por los jurados el 21 de Agosto del 2018, establece la fecha de sustentación, habiendo para el efecto cumplido los requisitos establecidos en el reglamento de para la Obtención del Grado Académico de Magister/Maestro y Doctor de la Escuela de Posgrado de la UANCV;

Que, en el Artículo 66 del Reglamento General de la Escuela de Posgrado de la UANCV, establece que la sustentación de Tesis de Postgrado es un trabajo de investigación original y crítico, de actualidad y de alto valor científico.

En uso de las atribuciones conferidas a la Dirección en el inciso "J" del artículo 17° del Reglamento General de la Escuela de Posgrado, y el Art. 74 del Estatuto Universitario;

SE RESUELVE:

ARTÍCULO PRIMERO.- NOMBRAR a los miembros del Jurado que calificarán la sustentación de la tesis del (a) Bach. ARROYO PAZ ANTONIO, con número de DNI. 01213151 y con número de matrícula 23120005, de la Maestría en INGENIERÍA DE SISTEMAS, Mención: INGENIERÍA DE SOFTWARE, de la Escuela de Posgrado de la Universidad Andina Néstor Cáceres Velásquez de Juliaca; quien ha presentado el Dictamen de Tesis: ESTUDIO COMPARATIVO DE PLATAFORMAS MIDDLEWARE OPEN-SOURCE DE INTERNET DE LAS COSAS (IoT) BASADO EN MÉTRICAS CUANTITATIVAS Y CUALITATIVAS. Nominado como ASESOR el (a) Dra. KATTY AGRIPINA PEREZ ORDOÑEZ, y siendo los jurados los siguientes docentes:

Presidente	:	MSc.	JUAN CARLOS HERRERA MIRANDA
Primer Miembro	:	Mgtr.	ALCIDES VELASQUEZ ARI
Segundo Miembro	:	Mgtr.	HENRY OSCAR VALENCIA GAMARRA

ARTÍCULO SEGUNDO.- DETERMINAR que la fecha de sustentación de Tesis, que se llevará a cabo fijando el siguiente lugar, fecha y hora.

Fecha	:	Viernes 19 de Octubre del 2018
Hora	:	16:00 p.m.
Local	:	Aula 202 Escuela de Posgrado - UANCV – JULIACA

A cuya finalización el Jurado registrará los resultados en el Libro de Actas de Sustentación de Tesis de Maestría con el grado de MAGISTER a los estudiantes que ingresaron anterior a la aprobación de la ley Universitaria N° 30220.

ARTÍCULO TERCERO.- ELEVAR la presente Resolución al Rectorado, Vicerrectorado Académico, Vicerrectorado Administrativo y Oficina del Órgano de Inspección y Control para conocimiento.

Regístrese, comuníquese y Archívese.

Cc./Archv.EPG (01)
Interesado (01)
Cargo (01)
Jurados (03)
Asesor (01)
Expediente (01)
OCM/agy

UNIVERSIDAD ANDINA NÉSTOR CÁCERES VELÁSQUEZ
ESCUELA DE POSGRADO
DIRECCIÓN
Dr. Cesar Augusto Colombres Mendia
DIRECTOR

UNIVERSIDAD ANDINA NÉSTOR CÁCERES VELÁSQUEZ
ESCUELA DE POSGRADO
SECRETARIO ACADÉMICO
Mg. Lara Chayña Aguilar
SECRETARIO ACADÉMICO

A mi amada esposa Nancy, por su paciencia y fortaleza, por impulsarme a concluir la tesis, a mis hijos Fairuz y César Antonio, con el deseo que mis huellas sean ejemplo de superación para ellos en sus vidas personales y profesionales.

A mis padres Nicólas y Olinda, que me brindaron su apoyo y fueron un motor, en la culminación de este trabajo, a mi hermano César que, desde Chile, me alentó.

Agradezco a Dios, ya que sin él no hubiera sido posible.

A la Escuela de Posgrado de la Universidad por permitirme concluir mis estudios y este trabajo de investigación.

A los honorables miembros del jurado MSc. Juan Carlos Herrera Miranda, Mgtr. Alcides Velásquez Ari, Mgtr. Henry Óscar Valencia Gamarra, a mi asesora Dr. Katty Agripina Pérez Ordoñez.

A mi gran amigo, el Dr. César Beltrán Castañón, que fue mi fortaleza para poder culminar la Tesis.

A Mauro Alexandre Amaro da Cruz, desde Brasil, por su apoyo recibido en la realización de este trabajo de investigación.

Í N D I C E

CAPÍTULO III

METODOLOGÍA DE LA INVESTIGACIÓN

CAPÍTULO IV

RESULTADOS Y PRUEBAS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El Internet de las cosas (IoT) es considerado una tecnología emergente compuesta por los sensores y actuadores que envían información a través de diferentes protocolos de comunicación conectados a puertas de enlace y plataformas middleware propietarias y de código abierto, estás últimas tienen un notable incremento.

El objetivo de este trabajo es realizar un estudio de evaluación del rendimiento de middleware open-source de IoT utilizando métricas cualitativas y cuantitativas. Se ha realizado una clasificación de veinte middlewares IoT open-source, tomando como muestra a dos de ellas, Konker y Thingsboard.

Se ha utilizado el framework SMI (Service Measurement Index) propuesto por el Cloud Services Measurement Initiative Consortium (CSMIC) para poder clasificar las métricas dentro de las categorías del SMI, en el caso de las métricas cualitativas se ha dado una escala de clasificación, y en las métricas cuantitativas se han tenido las siguientes: latencia medida en milisegundos, bytes enviados, bytes recibidos, medidos en bytes, tiempo de conexión medido en milisegundos, disponibilidad medida en porcentaje y costo medido en unidad monetaria. Se realizó pruebas en tiempo real de rendimiento con parámetros y número de usuarios usando el software Apache JMeter, directamente a dos VPS que se alquiló e instaló los middlewares, estos VPS están ubicados en Canadá perteneciente a la empresa OVH.

Los resultados obtenidos nos indica que las métricas cualitativas están sujetas a la percepción del usuario en cambio de las pruebas cuantitativas el middleware

thingsboard, es el que tiene la menor cantidad de bytes enviados y recibidos en comparación de konker, concluimos que el número de parámetros no incrementa significativamente los bytes enviados, en la métrica tiempo de conexión no se incrementa el valor de milisegundos al aumentar el número de parámetros, además encontramos que el número de usuarios simultáneos que acceden al middleware es de 500, como valor óptimo, teniendo que el middleware konker no presenta errores en comparación de Thingsboard que presenta un 18%, siendo un valor aceptable y en cuanto a la métrica de la latencia tiene un comportamiento menor hasta los 500 usuarios.

Palabras clave: internet de las cosas (Iot), plataformas middleware open-source, métricas cuantitativas y cualitativas

ABSTRACT

The Internet of Things (IoT) is considered an emergent technology composed of sensors and actuators that send information through different communication protocols connected to gateways and proprietary middleware platforms and code Open, they're late have a noticeable increase.

The research goal is to conduct a performance evaluation study of IoT open-source middleware using qualitative and quantitative metrics. It has made a classification of twenty IoT open-source middleware, taking as a sample two of them, Konker and Thingsboard.

The SMI (Service Measurement Index) framework proposed by the Cloud Services measurement Initiative Consortium (CSMIC) has been used to classify metrics within SMI categories, in the case of qualitative metrics a scale of Classification, and in the quantitative metrics have had the following: measured latency in milliseconds, sent bytes, received bytes, measured in bytes, connection time measured in milliseconds, availability measured in percentage and cost measured in unit monetary. Real-time performance tests were performed with parameters and number of users using the Apache JMeter software, directly to two VPS who rented and installed the Middleware, these VPs are located in Canada belonging to the OVH Company.

The results indicate that the qualitative metrics are subject to the perception of the user in exchange for quantitative tests Thingsboard middleware, is the one that has

the least number of bytes sent and received in comparison to Konker, we conclude that the number of parameters does not significantly increase the bytes sent, in the metric connection time does not increase the value of milliseconds by increasing the number of parameters, in addition we find that the number of simultaneous users that Access to the middleware is 500, as the optimal value, having the Konker middleware presents no errors in comparison with Thingsboard that presents 18%, being an acceptable value and in terms of the metric of latency has a lower behavior up to 500 users.

Keywords: Internet of Things (IoT), open-source middleware platforms, quantitative and qualitative metrics.

INTRODUCCIÓN

Acorde a la investigación sistemática y científica el estudio se ha esbozado en cuatro capítulos.

En el primer capítulo se presentan el problema, se realiza la exposición de la situación problemática, el objetivo general, los objetivos específicos a alcanzar,

En el segundo capítulo se muestran las bases teóricas del internet de las cosas, describiendo su arquitectura, se realiza un estudio del arte, presentando las diferentes plataformas open-source de internet de las cosas.

Se abordará todo lo que nos ofrece el internet de las cosas, presentamos un componente físico denominado nodemcu ESP8266-12E, que conectado al sensor de humedad y temperatura DHT-11, se encargan de recopilar la información y transmitir a una plataforma middleware, que almacena, procesa y muestra los resultados, mediante dashboard (paneles), que son visualizados por el usuario, en pantallas de computador, laptop, tablet o smartphone.

En este capítulo se analizan veinte plataformas middleware de internet de las cosas que tienen las características que son de código abierto (open-source), estos proyectos están alojados en el repositorio conocido como github, las clasificamos, estas plataformas middleware en un cuadro comparativo mostrando características que nos ayudan a poder distinguir unas de otras.

Para poder evaluar las plataformas middleware de internet de las cosas, necesitamos de una clasificación, para lo cual usamos el índice de medición del servicio(SMI), donde dividimos en dos categorías importantes, métricas cualitativas

y las métricas cuantitativas, las que nos ayudan a elegir la plataforma middleware adecuada son las métricas cuantitativas, al presentarnos valores que pueden cuantificar su medición.

En el tercer capítulo presentamos la metodología de investigación que es de tipo aplicativa, población, que realizando una búsqueda en internet encontramos veinte plataformas, tomamos una muestra de dos plataformas, para poder evaluarlas mediante las métricas cuantitativas, teniendo en cuenta que no todas las plataformas, se pueden instalar fácilmente, muchas de ellas solo tienen el código fuente y adolecen de la documentación, foros, blog, comunidad de usuarios, entre otros.

En el cuarto capítulo se muestran los procesos de evaluación de las métricas cuantitativas, ejecutando pruebas, para lo cual se hace uso del software de código abierto denominado Apache JMeter, que está construido en el lenguaje de programación Java, y que nos permitirá realizar pruebas de evaluación a las plataformas konker y thingsboard, referente a byte enviados y bytes recibidos, porcentaje de error, latencia.

Estas pruebas son realizadas en un escenario real, para cual se ha contratado el servicio de la empresa OVH, para poder obtener dos servidores privados virtuales (VPS), que se encuentran físicamente en Canadá.

Finalmente presentamos los resultados y recomendaciones.

CAPÍTULO I

EL PROBLEMA

1.1. EXPOSICIÓN DE LA SITUACIÓN PROBLEMÁTICA

Nos encontramos en un momento en que la conectividad se percibe en nuestro quehacer diario, El internet lo encontramos en el Hogar, Colegio, Oficinas, Negocios, Centros comerciales, entidades gubernamentales, no es ajeno encontrar acceso a Internet en estos lugares, y todo esto se ha trasladado a las cosas, que nos referimos a componentes físicos y virtuales, acompañados con una red de sensores.

Escuchamos hablar del termino Edificios Inteligentes, Ciudades Inteligentes (Smart cities), Autos sin conductor, entre otros, pero todo esto es gracias a una gran red de sensores que se conectan y están reportando información de sus actividades a servidores locales, o en la nube, estos sensores pueden ser de temperatura, gases, infrarrojos, presión arterial, latidos del corazón, calidad de agua, que son muy diversos.

A su vez también en las Industrias escuchamos hablar del termino de Industria 4.0, cuarta revolución industrial, que no es más el internet de las cosas dentro de la Industria.

Así mismo podemos destacar que han surgido con el advenimiento del IoT el desarrollo de plataformas Middleware tratando de dar un soporte a la cantidad de dispositivos heterogéneos, tipos de redes, protocolos y lenguajes de programación. El mercado de las plataformas IoT está en auge y en continua expansión de hecho hay encuestas que hablan que más del 80% de las empresas cree que el campo del IoT es el más interesante para sus negocios.

Otro método inalámbrico para acceder a Internet es a través de redes 3G, 4G y 5G. Ambos tienen los mismos problemas que Wi-Fi. Por esta razón, se desarrollaron soluciones inalámbricas de red de largo alcance como Sigfox, LoRa e IEEE 802.11 ah (HaLow). Como su nombre indica, consumen menos batería y proporcionan una amplia cobertura de área. Tanto LoRa como Sigfox necesitan una puerta de enlace que interactúe con los dispositivos finales. Esta puerta de enlace se conecta a una red de retorno que proporciona una conexión a Internet. Una de las diferencias entre LoRa y Sigfox es que Sigfox funciona de manera similar a un ISP tradicional, donde el usuario debe suscribirse al servicio para usarlo, mientras que LoRa ofrece tecnología que cualquier usuario puede comprar, instalar la infraestructura y usar la red a voluntad. La ventaja de IEEE 802.11ah sobre el resto de LoRa y Sigfox es que, como estándar IEEE 802.11, es compatible nativamente con redes IP. Otro método prometedor para acceder a Internet a través de IoT es la tecnología 5G, que se espera sea lanzada al público alrededor de 2020.

Últimamente se han fabricado computadores de placa reducida (en inglés: Single Board Computer o SBC) es una computadora completa en un sólo circuito. El diseño se centra en un sólo microprocesador con la RAM, E/S y todas las demás características de un computador funcional en una sola tarjeta que suele ser de tamaño reducido, y que tiene todo lo que necesita en la placa base. La característica principal de este hardware es su bajo costo debajo de los US$ 100.00, haciendo

posible su uso por su tamaño y cada vez con mayor capacidad de procesamiento, podemos destacar la placa Raspberry Pi 3 B+ que acaba de salir al mercado el mes de abril del 2018, teniendo un procesador de 64bits y una memoria Ram de 1Gb.

En el campo de la Ingeniería de Software nos encontramos con los Middleware que es la capa intermedia entre el hardware y las aplicaciones, es necesario evaluarla las diferentes plataformas middleware tanto propietarios como de código abierto, a su vez, es necesario poder evaluar cada uno de ellos que involucra una tarea bastante ardua debido a los diferentes componentes y escenarios que se desarrolla el Internet de las Cosas.

Las plataformas de desarrollo de aplicaciones se centran en desarrollar aplicaciones seguras que puedan escalar a muchos usuarios y ocuparse de la heterogeneidad presente en los entornos de IoT. Este tipo de plataformas también ofrece herramientas integradas para integrarse con proveedores de servicios populares, lo que permite que las aplicaciones desarrolladas sean compatibles con ellas

1.2. FORMULACIÓN DEL PROBLEMA

1.2.1 Interrogante principal

¿Cómo se puede elegir una plataforma middleware de IoT a través de métricas cualitativas y cuantitativas?

1.3 JUSTIFICACIÓN DE LA INVESTIGACIÓN

Para nuestra sociedad el Internet de las cosas aporta un valor importante, brindando información a muchas áreas, en el caso de una ciudad inteligente, temperatura ambiental, índice de contaminación sonora, trafico, semáforos conectados a sensores que pueden viabilizar las avenidas, el ahorro consumo de

energía en los postes de alumbrado eléctrico, son variables que ayudan a tomar decisiones, optimizar el presupuesto, poder mejor la ciudad entre otros aspectos.

En el área de salud, nos proporciona monitoreo de pacientes, pulsaciones por minuto, ritmo cardiaco, alarmas de estados críticos, los hospitales al implementar esta tecnología se benefician de contar con información valiosa que es de mucha ayuda.

En la actualidad se están desarrollando diversas plataformas middleware, tanto propietarias como open-source, funcionando con diferentes protocolos de comunicación, en forma local en servidores propios y en la nube, es importe el estudio de estas plataformas, clasificarlas, darles una valoración, para poder determinar cuál de ellas se adapta mejor a lo que queremos aplicar.

De igual manera, así como hay beneficios, también existen desafíos. En el momento en que un objeto se vuelve parte de un entorno interconectado, es necesario considerar que estos dispositivos han perdido seguridad física o lógica, ya que posiblemente se encontrarán localizados en entornos inhóspitos y serán accesibles al instante por individuos con malas intenciones. Los atacantes podrían interceptar, leer o modificar información, y podrían manipular sistemas de control y modificar la funcionalidad.

El aportar con un análisis del futuro del internet de las cosas y exponer las consideraciones sobre la seguridad que involucra que todos los dispositivos estén conectados en el internet, es difícil dimensionar la cantidad de situaciones a las cuales nos expondremos.

Se observa la necesidad y la importancia de realizar la investigación propuesta, ya que el internet de las cosas es una de las innovaciones tecnológicas que está en un proceso de incursión y que requiere de cuidado analizar las plataformas Middleware que brindara o que puede brindar esta tendencia tecnológica a los usuarios, instituciones o empresas.

Consideramos que los aportes a los que lleguemos serán útiles para que otros investigadores profundicen en otros trabajos relacionados al nuestro.

1.4 OBJETIVOS

1.4.1 Objetivo general

Realizar un estudio de evaluación del rendimiento de plataformas middleware open-source de Internet de las Cosas (IoT) utilizando métricas cuantitativas y cualitativas.

1.4.2 Objetivos específicos

- Caracterizar la situación actual de las plataformas middleware open-source.
- Utilizar el Índice de Medición del Servicio (SMI) para clasificar las métricas cuantitativas y cualitativas.
- Identificar las métricas cuantitativas para un middleware IoT.
- Identificar las métricas cualitativas para un middleware IoT.
- Utilizar el software Apache Jmeter como herramienta de prueba.
- Identificar que middleware IoT Open-source se desempeña mejor.

CAPÍTULO II

2. MARCO TEÓRICO

2.1 ANTECEDENTES DE LA INVESTIGACIÓN

En la presente investigación hemos realizado la búsqueda en diversas bibliotecas, bibliotecas digitales, repositorios y hemos encontrado los siguientes resultados

a. **J. Cardoso, C. Pereira, A. Aguiar, R. Morla. "Benchmarking IoT Middleware Platforms". 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM).**[1]

Se propusieron un conjunto de dimensiones cualitativas y métricas cuantitativas que se pueden utilizar para el benchmarking del middleware de IoT. Se usaron la publicación-suscripción de un gran conjunto de datos como caso de uso inspirado en un escenario de ciudad inteligente. La metodología comparo sistemáticamente las dos plataformas de middleware FIWARE y ETSI M2M.

Las dimensiones cualitativas fueron:

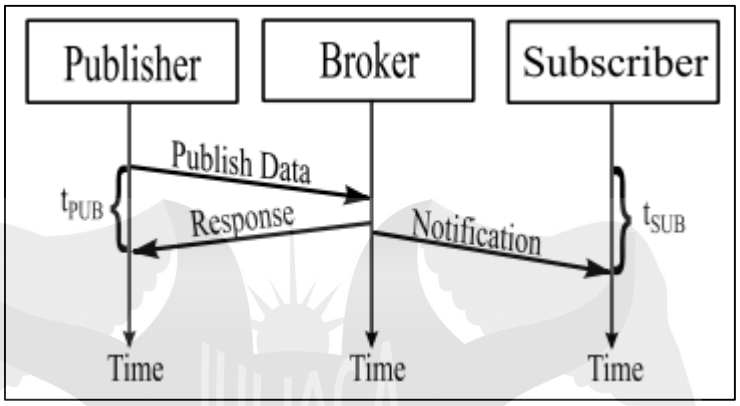
- soporte para el modelo de comunicación deseado (pub-sub, solicitud-respuesta, ...)

- requisitos de la aplicación IoT, basados en los requisitos definidos por el IoT-A;
- disponibilidad y claridad de la documentación, así como tutoriales disponibles;
- calidad del soporte y sustento de las comunidades de desarrolladores, por ejemplo, Stack Overflow.

Las dimensiones cuantitativas fueron:

- tiempo de publicación: tiempo transcurrido entre el envío de la solicitud HTTP de publicación y la recepción de la respuesta HTTP de publicación; t_{PUB} en la Figura 1;
- tiempo de suscripción: tiempo transcurrido entre el envío de la publicación Solicitud de HTTP y recibir la notificación de suscripción; t_{SUB} en la Figura 1;
- tiempo total: tiempo transcurrido entre el envío de la primera solicitud HTTP de publicación y la recepción de la última respuesta HTTP publicada;
- tamaño de los datos agrupados: gastos generales de serialización de datos medidos como el encabezado de longitud de contenido del protocolo HTTP, en bytes;
- tamaño de la publicación: tamaño en bytes de la carga útil TCP del paquete HTTP de publicación;
- cantidad total de datos utilizados para publicar un recurso: medida como la suma de los tamaños de nivel de red de todos los paquetes intercambiados (editor a broker y dirección inversa), en bytes;
- goodput: el número útil de bytes enviados en un período de tiempo determinado, medido como el tamaño de los datos serializados dividido por el tiempo de publicación.

Figura 1 Diagrama de secuencia que muestra los tiempos de publicación y suscripción.



Fuente: [1]

En el trabajo se indica que, durante el análisis de los resultados de rendimiento, se definieron un conjunto adicional de métricas que fueron útil para validar los resultados y proporcionar soporte para explicar los comportamientos observados en las principales métricas. Estas medidas auxiliares son:

- número de reintentos de solicitud HTTP, en los casos en que falla la solicitud HTTP de publicación debido a un problema con el intermediario, que da cuenta de errores en el código de estado HTTP de rango 500 o donde se envía el indicador TCP RST;
- tiempo de ida y vuelta al intermediario, obtenido a través de la diferencia entre la marca de tiempo del paquete TCP SYN y su respuesta;
- número de retransmisiones de paquetes TCP y HTTP y el retraso verificado para la retransmisión.

La metodología consiste en publicar un conjunto grande de datos en forma secuencial o en paralelo.

Para la publicación secuencial, realizaron 4 ciclos de publicación ya que la cantidad de tiempo requerida para realizar cada uno limitaba el número total de mediciones. Para la publicación paralela, cada experimento consistió en 10 ciclos de publicación con un número de solicitudes paralelas que van de 50 a 500, en incrementos de 50. Se limitaron el número de solicitudes paralelas a 500 porque los intermediarios no pudieron hacer frente a un mayor paralelismo peticiones.

Se concluye que las diferencias significativas entre las dos plataformas de middleware observadas en la publicación paralela se debieron principalmente a las limitaciones de la implementación de ETSI M2M. Por otro lado, también se observó variaciones de rendimiento a lo largo del día en FIWARE. También se concluyó que la publicación paralela puede reducir significativamente el tiempo total requerido para publicar todos los datos en comparación con la publicación secuencial.

Por ejemplo, el proxy de autenticación de FIWARE arrojó errores de HTTP en el rango de 500, cuando supuestamente no debería hacerlo de acuerdo con el equipo de FIWARE. El middleware ETSI M2M utilizado tenía varios problemas de implementación, a saber: las notificaciones de suscripción siempre se enviaban antes de enviar la respuesta HTTP OK a la solicitud de publicación, lo que puede crear un problema de escalabilidad en casos de gran cantidad de suscripciones para un recurso determinado.[1]

b. **Amaro da Cruz, Mauro Alexandre. (2017). "Performance Evaluation of IoT Middleware", Tesis de Maestría, Instituto Nacional de Telecomunicaciones – INATEL. Brazil. [2]**

A lo largo de esta disertación, se presentó un estudio actualizado sobre el middleware IoT, y se estudiaron y evaluaron el rendimiento de las soluciones de código abierto de middleware, así como una solución middleware patentada de Inatel (Instituto Nacional de Telecomunicaciones).

La disertación introdujo primero la motivación y delimitó el tema de investigación, describiendo los objetivos y mostrando sus principales contribuciones. También se presenta una tabla que muestra una lista de plataformas IoT y a qué categorías se dirige cada una de ellas. Luego, la disertación se centra en las plataformas de middleware IoT, detallando su propósito en IoT y cómo logran sus objetivos, a la vez que brinda más información sobre algunas de las soluciones de middleware existentes. El capítulo finaliza al proponer una arquitectura de referencia para soluciones de middleware que detalla el mejor método de operación de cada módulo.

Los 5 middleware que cumplían con el escenario propuesto procedieron a una evaluación de desempeño utilizando las métricas cuantitativas donde se verificó que cuando hay pocos usuarios simultáneos (hasta 100), la diferencia entre las soluciones es mínima, y no importa mucho qué middleware está implementado. La única excepción a esta regla es con 100 parámetros, en ese caso, el tiempo de respuesta de Linksmart es casi 12 veces más que el resto. En general, Orion + STH y Sitewhere fueron más estables en todos los experimentos. Sin embargo, no existe el mejor middleware, y cuando se implementa una solución de IoT, los usuarios deben tener en cuenta su escenario. En el caso de bajo rendimiento donde el tamaño del paquete es el aspecto más crucial, Konker y Linksmart son los mejores. Si el número de parámetros enviados por el dispositivo es el aspecto más crítico, Linksmart puede ser el mejor en la mayoría de los escenarios con 1 parámetro, o peor con 100 parámetros Si el porcentaje de error es la máxima prioridad, Orion y Sitewhere son los mejores con una tasa de error inferior al 1% (Orion es un poco mejor). Si el número de usuarios concurrentes no es más de 5000, InatelPlat y Konker también son viables.

Las métricas relacionadas con el tamaño del paquete no deben subestimarse, ya que es una herramienta valiosa para dimensionar la solución. Imagine que los dispositivos están conectados directamente a una puerta de enlace, que luego reenvía las solicitudes al middleware, y Konker (el tamaño de paquete más eficiente) es el middleware que se utiliza, también, el objetivo es enviar 15 parámetros. La velocidad de transferencia de la puerta de enlace es de 1 Mbps. Ahora considere que los paquetes se envían a la puerta de enlace sin compresión, lo que significa que se envían 535 Bytes y se reciben 580. La cantidad máxima de dispositivos por puerta de enlace será 215 (porque se reciben más bytes que enviados en este caso). En un entorno de IoT real, los datos enviados a la puerta de enlace se comprimirían. Sin embargo, en un entorno de IoT real, es común que los dispositivos se conecten a través de una red de malla que, por lo general, es más lenta y muy concurrida, lo que significa que se conectarán menos dispositivos directamente a la puerta de enlace. Conocer el tamaño del paquete que se transmite en cada

escenario ayuda a los usuarios a distribuir la carga de la red y permite planificar su solución de IoT.

c. **V. Araujo Soto. "Performance evaluation of scalable and distributed IoT platforms for smart regions". 2017. Master's Thesis. Luleå University of Technology. Suecia. [3]**

En este trabajo se evalúan el rendimiento de los componentes de la Arquitectura de Habilitación de Servicios de IoT de la iniciativa europea FIWARE. Se desarrolla un banco de pruebas que puede inyectar actualizaciones de datos usando MQTT y los protocolos Lightweight M2M basados en CoAP, simulando despliegues de IoT a gran escala. Las pruebas consideraron la escalabilidad vertical y horizontal de los componentes de la plataforma.

Esta tesis ofrece una evaluación del rendimiento de los agentes de IoT y los componentes de Orion Context Broker que se definen en la arquitectura de habilitación de servicios de IoT de FIWARE. Esta evaluación de rendimiento consideró el rendimiento alcanzado, la utilización de recursos computacionales y latencias para consultas de entidades a la interfaz hacia el norte. Las evaluaciones de desempeño también consideraron las diferentes estrategias que se pueden usar para escalar los componentes de la plataforma y sus funcionalidades generales en la nube, para acomodar los datos que se envían activamente desde un número cada vez mayor de dispositivos en Internet of Things.
En primer lugar, el diseño de banco de prueba admite los protocolos MQTT y LWM2M IoT, no se habían probado previamente. En segundo lugar, nuestra generación distribuida de solicitudes asíncronas creó las condiciones de estrés necesarias para observar el comportamiento y las limitaciones de los componentes FIWARE estudiados bajo carga. Las limitaciones metodológicas en trabajos previos no permitieron observar estos límites. Finalmente, nuestro trabajo consideró la capacidad de los componentes para manejar más carga a través de escalas verticales u horizontales, dando una caracterización más amplia de las capacidades

y el potencial de la plataforma para servir a escenarios a gran escala, en comparación con el trabajo previo.

Los resultados permiten hacer recomendaciones para implementadores de FIWARE en ciudades inteligentes u otros entornos. Para niveles de carga más bajos (menos de 1000 solicitudes de actualización de datos por segundo, para todas nuestras configuraciones estudiadas), escalar verticalmente no es una estrategia rentable. Escalar Orion horizontalmente detrás de un equilibrador de carga tampoco admitía muchas más solicitudes por segundo. De los resultados, los implementadores de aplicaciones inteligentes para ciudades inteligentes pueden confiar en nuestra configuración básica para niveles bajos de carga. Si es necesario escalar de forma sólida, los esfuerzos deben centrarse en la implementación de un clúster fragmentado apropiado para la base de datos. Al mismo tiempo, la implementación de FIWARE de agentes de IoT debe mejorarse para soportar la carga de escenarios actuales y futuros a gran escala para ciudades inteligentes. Señalamos su uso anormal de la memoria y las conexiones TCP bajo alta carga, y las limitaciones arquitectónicas de utilizar un solo núcleo para el procesamiento.

Nuestro trabajo proporciona otras contribuciones que se pueden utilizar más allá de FIWARE.

Finalmente, a través de las lecciones aprendidas sobre la forma de implementar componentes de la plataforma FIWARE para soportar más carga, este trabajo ayuda a acercar la realidad a la visión IoT para ciudades inteligentes y sus beneficios de sostenibilidad.

Los resultados de este trabajo también pueden ser utilizados por los desarrolladores para optimizar las implementaciones actuales y corregir los componentes que actúan de forma inestable bajo carga.

d. **A. Salami, A. Yari."A framework for comparing quantitative and qualitative criteria of IoT platforms". 2018. 4th International Conference on Web Research (ICWR).** [4]

En este trabajo se analizan tres plataformas, Thingspeak, Xively y AWS IoT, solo una de ellas, Xyvely es de código Abierto, las otras dos requieren de un pago por uso.

Se crean tres fuentes de datos temperatura, humedad e intensidad de luz para simular los escenarios de prueba mencionados utilizando las plataformas de IoT anteriores.

Para las pruebas se consideran 3 métricas

- Fiabilidad
- Latencia
- Escalabilidad

1. Comparación de plataformas basada en la fiabilidad

Las plataformas Xively y Thingspeak confrontaron una alta tasa de envío de información, es decir, enviando en menos intervalos de tiempo, muy pocos datos fueron recibidos correctamente en 20 minutos, y por lo tanto, eran menos confiables. Esto es particularmente evidente en intervalos de tiempo de uno y tres segundos, y su fiabilidad es casi la misma. pero a intervalos más altos La plataforma Xively está separada y relativamente ha recibido la cantidad de datos más alta que la plataforma Thingspeak, lo que ha resultado en un mayor grado de confiabilidad, aunque ambas plataformas tienen altibajos en el número de datos recibidos en intervalos de tiempo de 10 a 15 segundos. Pero se puede concluir que la plataforma Xively en estos intervalos es mejor resultado. La plataforma AWS IOT es diferente, porque incluso con la alta velocidad de envío de información, hay un porcentaje significativo de confiabilidad (más del 70%). y como el tiempo de transmisión de datos de los dispositivos se ralentiza, la pendiente del porcentaje de confiabilidad se eleva al 100%. Entonces, está claro que esta plataforma, en términos de evaluación de este indicador, es mucho mejor que las otras dos plataformas.

2. Comparación de plataformas basada en la latencia

De acuerdo con los resultados de varias pruebas realizadas, se puede concluir que la plataforma AWS IOT tiene una latencia promedio menor en la recepción de datos de cada dispositivo que en las otras dos plataformas, y la plataforma Xively también tiene un retraso levemente menor que l plataforma. Mientras que el promedio total de la latencia de los datos de recepción de los dispositivos por parte de la plataforma AWS IoT es mayor que el de Xively, y esta plataforma es superior a la de Thingspeak. Debido al alto porcentaje de su confiabilidad, a medida que recibe más cantidad de datos, se calcula una mayor cantidad de datos y una mayor latencia de los dispositivos para obtener el promedio.

Pero otro punto importante acerca de la eficiencia en términos de tiempo de retardo es obtener una desviación estándar para determinar la dispersión de la latencia de la plataforma para recibir información de dispositivos en diferentes momentos, de modo que, si la latencia de recibir información difiere considerablemente en momentos diferentes, reduce la eficiencia del sistema.

C. Comparación de plataformas basada en la escalabilidad

Como sabemos, uno de los puntos de referencia para evaluar cómo una plataforma es más escalable, es que la cantidad de dispersión de datos recibidos de los dispositivos debería acercarse a cero en un intervalo de tiempo dado. Desde esta perspectiva, la plataforma AWS IOT es mucho más escalable que las otras dos plataformas, porque, como se muestra en el diagrama, la desviación estándar de las otras dos plataformas en un número diferente de dispositivos está entre uno y dos, mientras que el resultado numérico en todo este número de dispositivos para la plataforma AWS IOT es menor a 0.5.

Pero otra cosa que debe considerarse sobre el benchmark de escalabilidad es la cantidad de dispersión de datos que crece cuando se agrega la cantidad de dispositivos. Si esta tasa de crecimiento es demasiado alta, significa que, al agregar la cantidad de dispositivos, la cantidad de dispersión cambiará repentinamente mucho, lo que ocasionará una disminución en el rendimiento de la plataforma.

En este trabajo no se indica que herramienta de prueba utilizaron.

2.2 BASES TEÓRICAS

2.2.1. Ingeniería de software para internet de las cosas

Las técnicas de ingeniería de software pasadas pueden aprovecharse y adaptarse a los desafíos del IoT actual. Sin embargo, también se necesitan nuevos enfoques para las técnicas de ingeniería de software estándar, por ejemplo, replantear la gestión de la configuración en el contexto de los sistemas extremadamente dinámicos y de reconfiguración continua que son característicos del IoT.

Se necesita una nueva generación de entornos de desarrollo para la ingeniería de software para el IoT. Una de las tendencias más interesantes son los entornos de desarrollo en la nube, para permitir las técnicas de validación y verificación masivamente escalables (incluida la simulación) que serán necesarias para la mayoría de los grandes sistemas de misión crítica en el IoT. Algunos proveedores de tecnología ahora están ofreciendo plataformas que permiten a los ingenieros de software ensamblar rápidamente sistemas que interactúan con sofisticados dispositivos de sensores, pero que conservan importantes propiedades a nivel de sistema.

La industria necesita orientación para diseñar la nueva generación de sistemas de software escalable, altamente reactivo, a menudo, de recursos limitados, característicos del IoT. Muchos de estos sistemas se encuentran en sectores de misión crítica, como la medicina, la automatización industrial y la gestión energética. Por encima de todo, tenemos que capacitar a la nueva generación de desarrolladores de software IoT. No podemos simplemente darles rienda suelta para desarrollar aplicaciones de IoT de una manera descontrolada y posiblemente poner en peligro vidas.[5]

2.2.2 Internet de las cosas

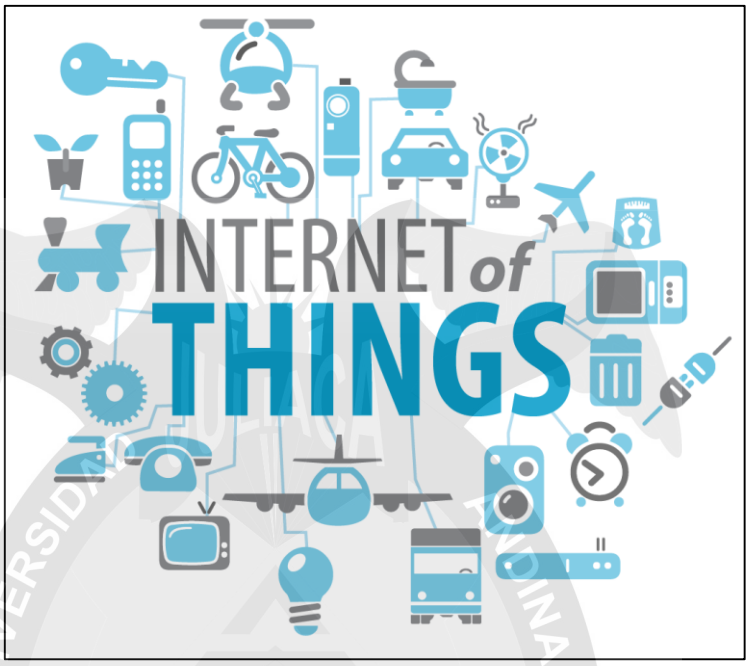
Para poder definir el termino de Internet de las Cosas, podemos referenciar al IEEE, que a través de un portal iot.ieee.org tienen una iniciativa de poder agrupar el conocimiento sobre el IoT.

El término "Internet of Things" (IoT) traducido al castellano Internet de las cosas, se le atribuye a Kevin Ashton ya que, en 1999, comenzó una presentación titulada "That 'Internet of Things' Thing" [6]. para describir un sistema en el que Internet está conectado al mundo físico a través de sensores ubicuos. Como ubicuo nos referimos a que los sensores están en todas partes. Podemos decir que, a partir de entonces, se realizaron enormes contribuciones, como seguridad, conectividad, eficiencia energética y mucho más sobre el tema. Actualmente, IoT se considera un tema relevante para investigadores, consumidores y proveedores de servicios. Desde su comienzo, el término ha sufrido modificaciones mínimas.

La Unión Internacional de Telecomunicaciones define:

"IoT puede concebirse como una infraestructura global de la sociedad de la información, que permite ofrecer servicios avanzados mediante la interconexión de objetos (físicos y virtuales) gracias a la interoperatividad de tecnologías de la información y la comunicación (TIC) presentes y futuras." [7]

Figura 2 Internet de las Cosas



Fuente:[8]

2.2.3 Nodemcu

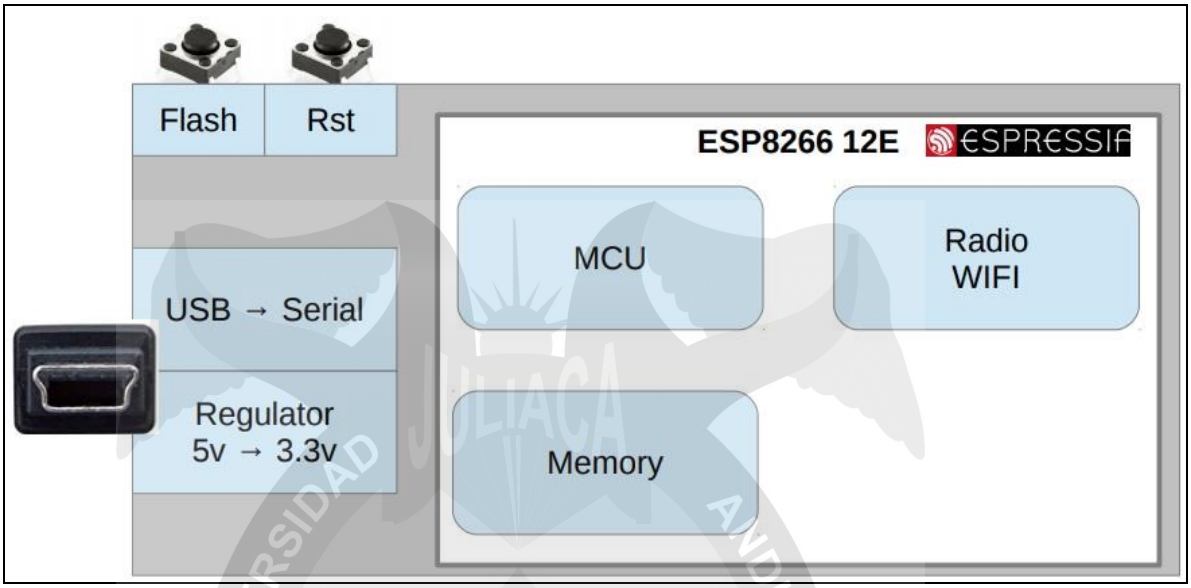
Es un microcontrolador de placa única, conformado por Hardware y Software, indicamos que el firmware está basado en Lenguaje Script Lua [9], que se ejecuta en el chip ESP8266 Wi-Fi SoC de la empresa Espressif Systems. [10]
MCU se refiere a MicroControlador (MicroController Unit) por sus siglas en inglés, tiene las dimensiones de 5mm x 5mm, es un chip que contiene procesador, RAM, ROM, reloj y la unidad de control de E/S en un único componente, es de bajo costo.

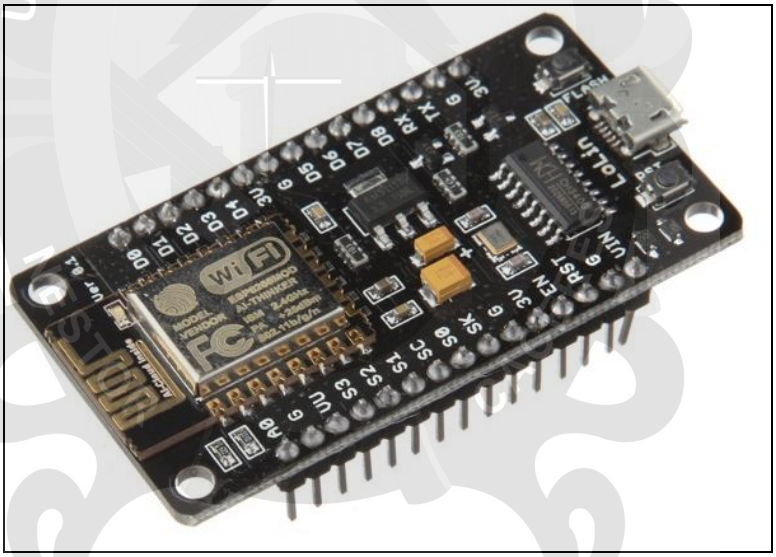
El NodeMcu, está compuesto por entrada micro Usb, donde se alimenta de 5v y se convierte a 3.3v, Botones Flash y Reset.

- botón Reset, permite reiniciar el programa
- Botón Flash, encargado de borrar la memoria y si hubiere algún programa grabado, este se elimina.

Tambien posee Wi-Fi. Conformado por una antena, para la transmisión y recepción.

Figura 3 Diagrama de Bloques del Nodemcu





Fuente: [11]

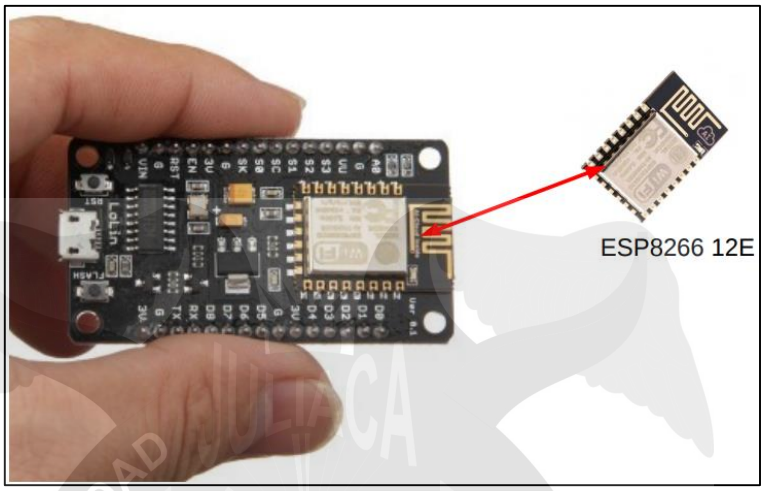
Principales Características

Tabla 1 Características del NodeMcu

Descripción	Características
Wireless Standard	IEEE 802.11 b/g/n
Frequency Range	2.412 - 2.484 GHz
Power Transmission	802.11b : +16 ± 2 dBm (at 11 Mbps) 802.11g : +14 ± 2 dBm (at 54 Mbps) 802.11n : +13 ± 2 dBM (at HT20, MCS7)
Receiving Sensitivity	802.11b : -93 dBm (at 11 Mbps, CCK) 802.11g : -85 dBm (at 54 Mbps, OFDM) 802.11n : -82 dBm (at HT20, MCS7)
Wireless Form	On-board PCB Antenna
IO Capability	UART, I2C, PWM, GPIO, 1 ADC
Características Eléctricas	3.3 V Operated 15 mA output current per GPIO pin 12 - 200 mA working current Less than 200 uA standby current
Operating Temperature	-40 to +125 ºC
Serial Transmission	110 - 921600 bps, TCP Client 5
Wireless Network Type	STA / AP / STA + AP
Tipo de Seguridad	WEP / WPA-PSK / WPA2-PSK
Tipo de Encriptación	WEP64 / WEP128 / TKIP / AES
Firmware Upgrade	Local Serial Port, OTA Remote Upgrade
Network Protocol	IPv4, TCP / UDP / FTP / HTTP
User Configuration	AT + Order Set, Web Android / iOS, Smart Link APP

Fuente: [10]

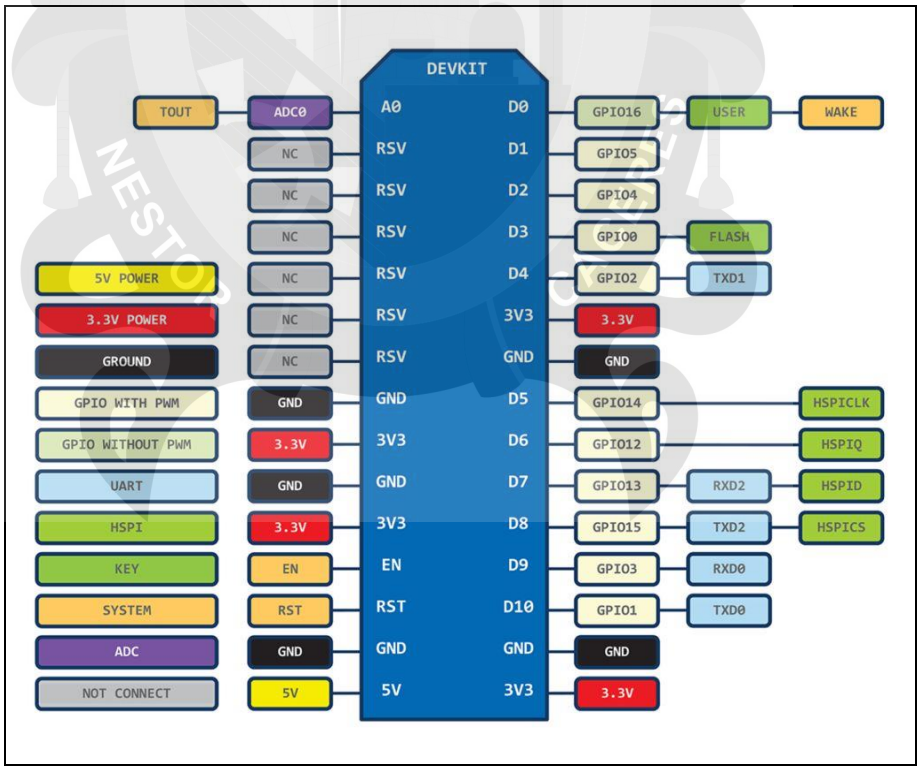
Figura 4 NodeMcu Conteniendo el Microcontrolador Esp8266-12E



Fuente [11]

El NodeMcu tienes una seria de pines que permiten la comunicación con los sensores, a su vez que proporcionan voltaje, punto de tierra, entrada digital y analógica, presentamos la figura donde se aprecia este el mapa de los pines.

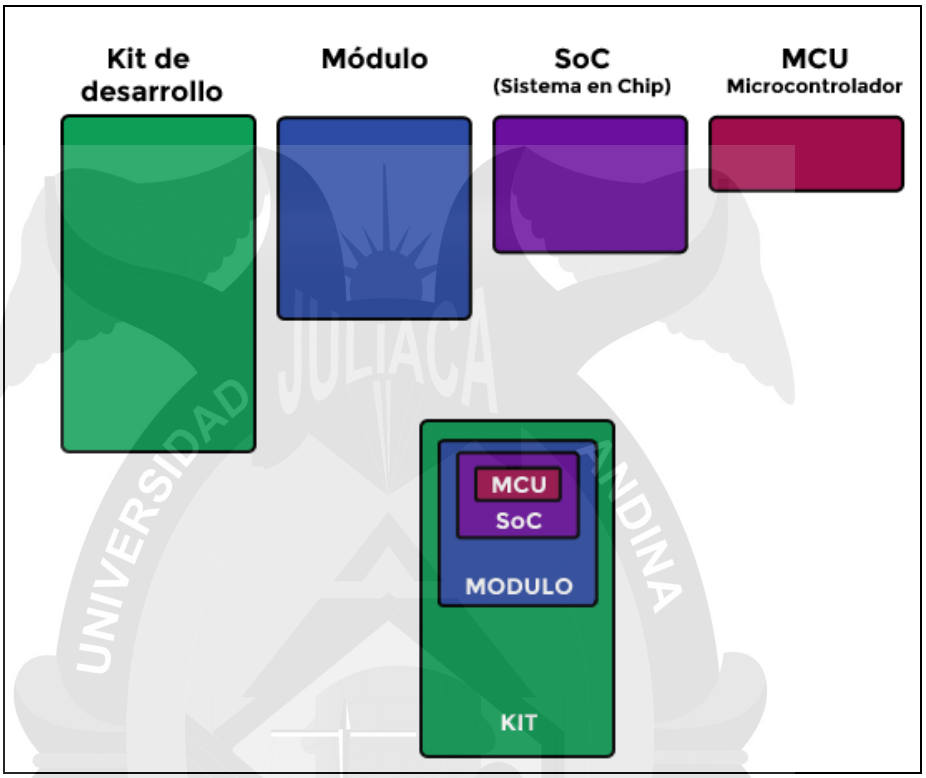
Figura 5 Pines del Nodemcu



Fuente [11]

Kit de Desarrollo

Figura 6 Kit de Desarrollo Del NodeMCU



Fuente:[12]

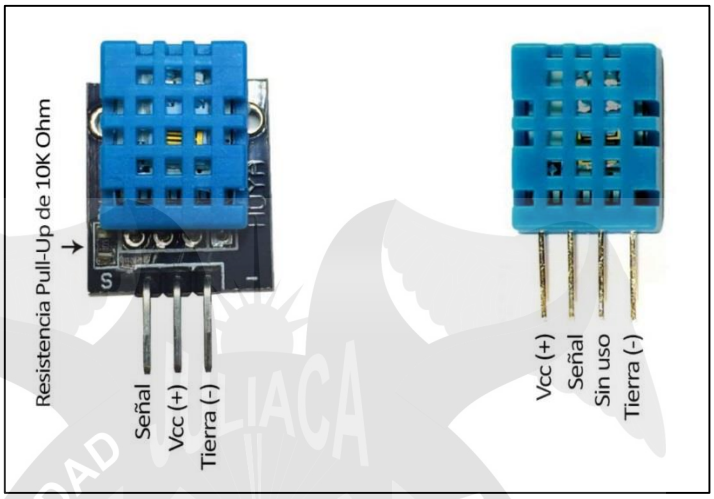
2.2.4 Sensor de temperatura y humedad DHT11

Está constituido por dos sensores resistivos (NTC y humedad). Tiene una excelente calidad y una respuesta rápida en las medidas. Puede medir la humedad entre el rango 20% aprox. 95% y la temperatura entre el rango 0ºC a 50ºC.

Cada sensor DHT11 está estrictamente calibrado en laboratorio, presentando una extrema precisión en la calibración. Los coeficientes de calibración se almacenan como programas en la memoria OTP, que son empleados por el proceso de detección de señal interna del sensor.

El protocolo de comunicación es a través de un único hilo (protocolo 1-wire), por lo tanto, hace que la integración de este sensor en nuestros proyectos sea rápida y sencilla. Además, presenta un tamaño reducido, un bajo consumo.[13]

Figura 7 Sensor DHT11 Humedad y Temperatura



Fuente: [14]

TABLA 2 CARACTERISTICAS DEL SENSOR DHT11

Características	Descripción
Alimentación	3Vdc ≤ Vcc ≤ 5Vdc
Corriente	0.2mA
Señal de salida	Digit
Rango de medida / temperatura	De 0 a 50
Precisión de temperatura	±2
Resolución de temperatura	0.1°
Rango de medida / humedad	De 20% a 90% RH
Precisión de humedad	4%
Resolución de humedad	1%R
Tamaño	12 x 15.5 x 5.5mm

Fuente: [14]

DHT11 se compone de un sensor de humedad resistivo y un termistor de coeficiente negativo de temperatura (NTC), ambos controlados a través de un chip de micro-controlador de 8 bits que se encuentra contenidos en una pequeña caja de plástico azul de 15mmx12mm. Existen varios modelos del sensor DHT11 en el mercado, algunos vienen soldados a una placa y otros vienen sueltos para usarse directamente conectados a un protoboard. La diferencia es que los modelos que

están soldados a una placa ya tienen una resistencia y nos evita el uso de estas. Podemos encontrar varios diagramas en Internet de conexión del DHT11 con resistencias. Para nuestro proyecto utilizamos el sensor DHT11 con placa, que tiene como salida 3 pines.

2.2.5 Arduino ide

Es el entorno de Desarrollo Integrado (IDE), el software mediante el cual podemos realizar la programación del NodeMCU, para el presente proyecto utilizamos la versión 1.8.7, que se puede descargar del https://www.arduino.cc/en/Main/Software en donde también encontramos versiones como el Arduino Web Editor, que es una versión Web. Podemos indicar que el Arduino IDE, viene para varios sistemas operativos, como Mac OS X, Linux y Linux ARM, este último diseñado para Raspberry Pi y compatibles. Es necesario tener instalado Java.

Figura 8 Arduino IDE

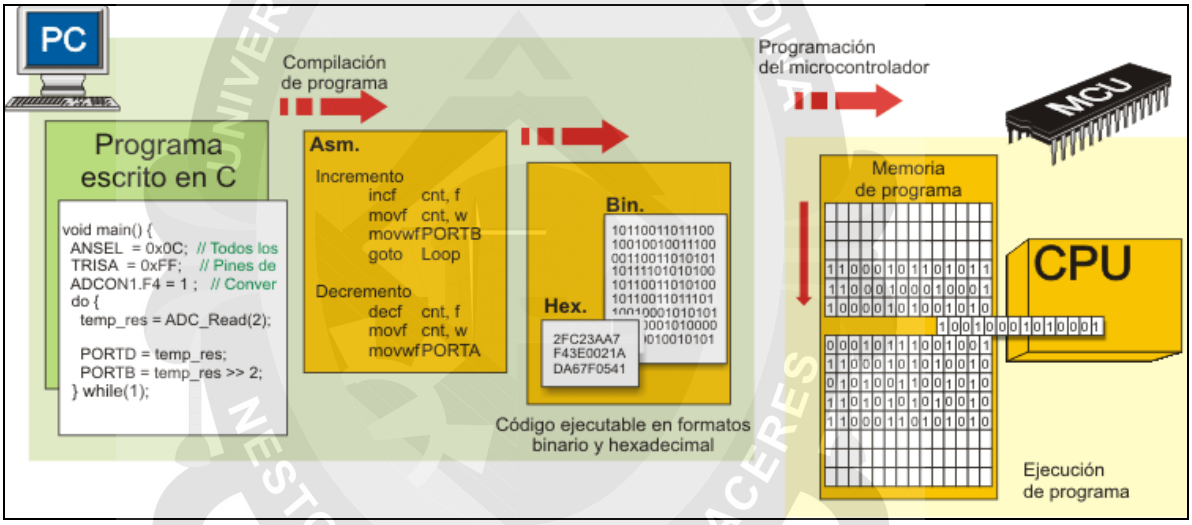
Fuente: [15]

Últimamente han aparecido IDE de programación a través de bloques parecidos a scratch, como son:

- http://www.visualino.net/
- http://www.mblock.cc/

El lenguaje de programación del Arduino IDE, es el lenguaje C, en el siguiente gráfico podemos ver el proceso que se lleva a cabo hasta la ejecución del programa en el Micro Controlador (MCU)

Figura 9 proceso de compilación del programa



Fuente: [16]

2.2.6 Open hardware

Hardware de Fuentes Abiertas (OSHW en inglés) es aquel hardware cuyo diseño se hace disponible públicamente para que cualquier persona lo pueda estudiar, modificar, distribuir, materializar y vender, tanto el original como otros objetos basados en ese diseño. Las fuentes del hardware (entendidas como los ficheros fuente) habrán de estar disponibles en un formato apropiado para poder realizar modificaciones sobre ellas. Idealmente, el hardware de fuentes abiertas utiliza componentes y materiales de alta disponibilidad, procesos estandarizados,

infraestructuras abiertas, contenidos sin restricciones, y herramientas de fuentes abiertas de cara a maximizar la habilidad de los individuos para materializar y usar el hardware. El hardware de fuentes abiertas da libertad de controlar la tecnología y al mismo tiempo compartir conocimientos y estimular la comercialización por medio del intercambio abierto de diseños.

El «Hardware de fuentes abiertas» es un término para denominar artefactos tangibles – máquinas, dispositivos, u otros objetos del mundo físico – cuyo diseño ha sido publicado de forma tal que cualquier persona pueda fabricar, modificar, distribuir y usar esos objetos. Esta definición tiene la intención de proveer una guía para el desarrollo y evaluación de licencias para «Hardware de fuentes abiertas».

Los términos de distribución del «Hardware de fuentes abiertas» habrán de seguir los siguientes criterios:

1. Documentación
2. Alcance
3. Programas informáticos necesarios
4. Obras derivadas
5. Libre redistribución
6. Atribución
7. No discriminación a personas o grupos
8. No discriminación a campos de aplicación
9. Distribución de la licencia
10. La licencia no será específica a un producto
11. La licencia no deberá restringir otro Hardware o Software
12. La licencia será neutra en términos tecnológicos

Figura 10 Open Hardware

Fuente: [17]

2.2.7 Open source

El hecho de que un programa se considere de código abierto no es solo para hacer que el código fuente esté disponible para todos. A continuación, se detallan los requisitos que se requieren para que un programa se considere de código abierto [18].

Figura 11 Open Source

Fuente: [18]

Requisitos:

1. La licencia del programa no impedirá que los usuarios vendan o eliminen componentes que contengan el programa dado y que la licencia no requerirá ningún tipo de tarifa cuando se venda un componente.

2. El programa debe incluir el código fuente y permitir la distribución en el código fuente, así como también en forma compilada. En caso de que un producto se distribuya sin el código fuente, debe publicarse una referencia clara acerca de cómo se obtiene el código fuente, esto por un costo razonable de reproducción, pero preferiblemente mediante descarga gratuita desde Internet. El código fuente debe ser la forma preferida en que un programador modifique el programa. Crear conscientemente un código fuente confuso no está permitido. Formas intermedias como la salida de un preprocesador o intérprete no está permitida.
3. La licencia del programa no debe causar ningún obstáculo en caso de que se realicen modificaciones en el programa.
4. La licencia del programa puede restringir la distribución del código fuente modificado, pero solo si la licencia permite la distribución de "archivos de parche" con el código fuente para modificar la aplicación en el momento de la construcción. La licencia solo puede permitir la distribución de programas creados con el código fuente modificado. La licencia puede requerir que los programas creados tengan un nombre o número de versión diferente al del software original.
5. La licencia del programa no puede discriminar ni ninguna ni ninguna.
6. La licencia del programa no debe limitar a nadie a crear un programa para un propósito específico. Por ejemplo, la licencia no puede limitar la investigación en un área específica.
7. Los derechos del programa deben aplicarse a todos los que el programa se redistribuya sin que se creen nuevas licencias para esas partes.
8. Los derechos del programa no deben de ninguna manera depender de que el software sea parte de una distribución de software específica. Si el programa se recupera de esa distribución y se usa o distribuye bajo los términos de la licencia del programa, todas las partes a las cuales se redistribuye el programa deben tener los mismos derechos que los otorgados en relación con la distribución original del software.

9. La licencia del programa no puede crear ninguna restricción sobre otro software distribuido junto con el software licenciado. Por ejemplo, la licencia puede no insistir en que cualquier otro software distribuido en el mismo medio debe ser de código abierto.
10. Ninguna disposición de la licencia puede basarse en ninguna tecnología específica o tipo de interfaz.

2.3. ESTADO DEL ARTE

2.3.1 Plataforma IoT

Para adoptar diferentes dispositivos de Internet of Thing y brindar un mejor servicio, las plataformas de IoT salen a gestionar los dispositivos y servicios conectados, el desarrollo de aplicaciones, almacenar e incluso analizar grandes cantidades de datos generados. La plataforma IoT tiene tres componentes principales: basado en la nube, conectado, impulsado por datos. Utilizando la plataforma Internet of Things, los desarrolladores pueden prestar más atención a las aplicaciones de valor agregado en lugar de a los niveles bajos de tecnología, lo que significa que las empresas pueden llevar los productos a los mercados con mayor rapidez. Las plataformas que se encuentran en el mercado generalmente se pueden clasificar en tres tipos: sistemas centralizados (o distribuidos en la nube), semi-distribuidos y totalmente distribuidos. [19]

(a) Sistemas centralizados

La mayoría de las plataformas de Internet de las cosas se basan en la nube y pueden considerarse centralizadas porque siempre transmiten la información del sensor y del actuador a través de un punto centralizado. Consulte en la Figura 11 la descripción general de la arquitectura centralizada. Los ejemplos típicos de la arquitectura incluyen: Thingworx, Kaa, ThingSpeak, Xively, Nimbits, etc.

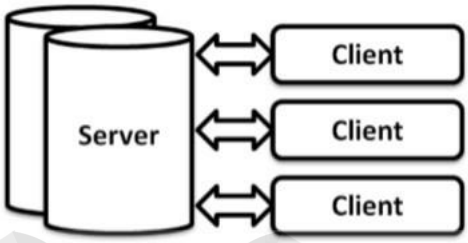


Figura 12 Descripción general de la arquitectura centralizada típica

(b) Sistemas semidistribuidos

Las plataformas semi-distribuidas de Internet de las cosas a menudo se basan en protocolos de iniciación de sesión, y generalmente contienen un punto centralizado para coordinar la comunicación. Por eso, corren más rápido y escalan más fácilmente que los sistemas centralizados. Consulte en la Figura 12 la descripción general de la arquitectura semi-distribuida. Los ejemplos típicos de eso incluyen: SENSEI, ETSI M2M y otras plataformas basadas en 3GPP IMS.

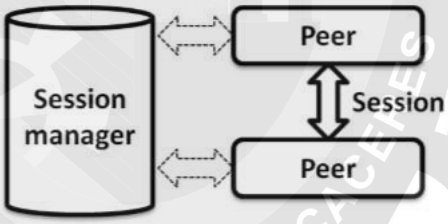


Figura 13 Descripción general de la arquitectura típica semi-distribuida

(c) Sistemas completamente distribuidos

Pocas plataformas de Internet de las cosas están distribuidas en su totalidad operan de una manera de igual a igual, donde los nodos pueden intercambiar información directamente. Consulte en la Figura 13 la descripción general de la arquitectura de distribución completa. Cada entidad necesita almacenar y administrar en el cliente peer-to-peer. Estos sistemas no contienen un único punto de falla y, por lo tanto, son más resistentes. Ejemplos típicos de eso incluyen: LinkSmart, Nabto y SensibleThings.

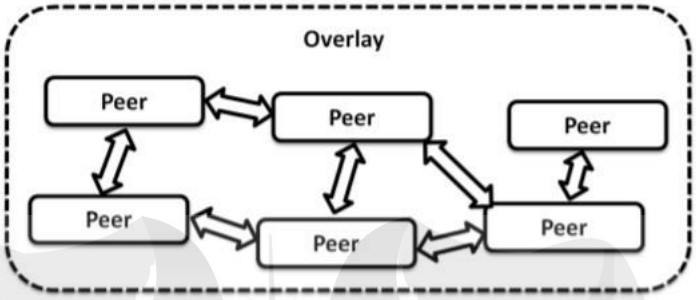


Figura 14 Descripción general de la arquitectura típica de distribución completa

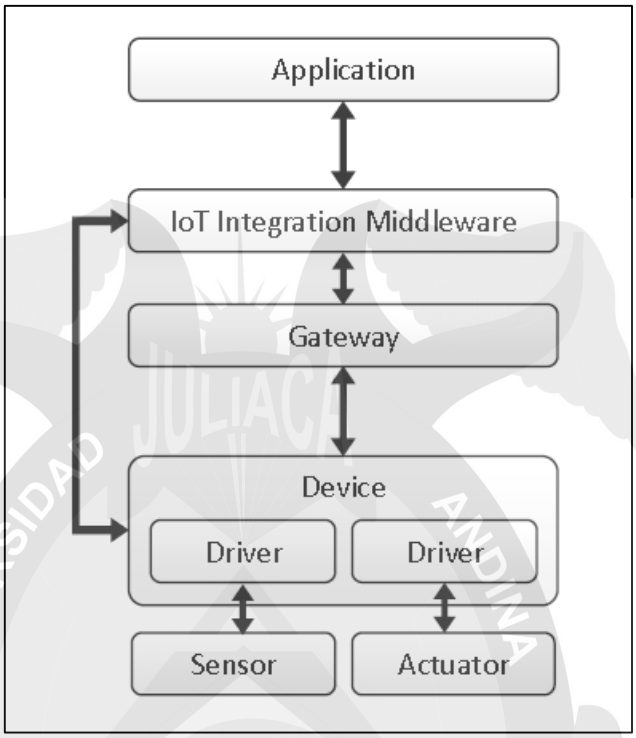
2.3.2 Arquitectura middleware

La arquitectura de referencia de IoT que se describe a continuación se deriva de una comparación de varias plataformas IoT, incluidas las de código abierto y propietarias, tomadas del Trabajo de [20] se presenta las arquitecturas IoT, compuesta por 5 capas, con sus componentes y su intercomunicación, según Figura 14.

En aras de la simplicidad, los componentes se representan sin cardinalidades. Además, los componentes también se pueden omitir.

Por ejemplo, si un sistema ciberfísico solo se usa para medir los parámetros del entorno físico, el sistema no tendría actuadores. En contraste con las arquitecturas de referencia existentes, mantuvimos nuestro resumen a propósito ya que el objetivo de nuestra arquitectura de referencia es servir como una terminología uniforme y abstracta, lo que facilita la comparación de diferentes plataformas. A continuación, describimos cada una de las capas.

Figura 15 Arquitectura de referencia de IoT



Fuente: [20]

2.3.3 Sensor

Un sensor es un componente de hardware que captura información en el entorno físico al "responder a un estímulo físico (como calor, luz, sonido, presión, magnetismo o un movimiento particular)". Por ejemplo, al medir la humedad dentro de una habitación, un sensor ubicado dentro de esa habitación captura el nivel de humedad de la habitación. Los sensores transmiten la información capturada mediante señales eléctricas a los dispositivos a los que están conectados. Esta conexión se puede establecer (i) por cable o (ii) de forma inalámbrica. La conexión por cable incluye una integración de sensores en un dispositivo. Un sensor se puede configurar mediante un software, pero no puede ejecutar software por sí mismo.

2.3.4 Actuador

Un actuador es un componente de hardware que manipula el entorno físico. Los actuadores reciben comandos de su dispositivo conectado y traducen estas señales eléctricas en algún tipo de acción física. Por ejemplo, un Actuador que enciende o apaga una ventilación dentro de una habitación actúa sobre el entorno físico al influir en la humedad de la habitación. Al igual que con los sensores, la conexión al dispositivo se puede establecer (i) por cable o (ii) de forma inalámbrica, por lo que una conexión por cable incluye la integración en un dispositivo. Además, un Actuador se puede configurar usando un software, pero no puede ejecutar el software por sí mismo.

2.3.5 Dispositivo

Un Dispositivo es un componente de hardware que (i) está conectado a Sensores y/o actuadores por cable o de forma inalámbrica o (ii) incluso integra estos componentes. Los dispositivos tienen un procesador y una capacidad de almacenamiento para ejecutar software y establecer una conexión con el Middleware de IoT. Por lo tanto, los dispositivos son el punto de entrada del entorno físico al mundo digital. Un controlador es un software que se ejecuta en el dispositivo que permite el acceso uniforme a sensores y actuadores heterogéneos. Los dispositivos son (i) autónomos o (ii) conectados a otro sistema más grande.

2.3.6 Gateway

En caso de que un Dispositivo no sea capaz de conectarse directamente a otros sistemas, está conectado a una Puerta de enlace. Un Gateway proporciona las tecnologías y los mecanismos necesarios para traducir entre diferentes protocolos, tecnologías de comunicación y formatos de carga útil. Transmite la comunicación entre los dispositivos y otros sistemas. Cuando la puerta de enlace recibe un mensaje en un formato binario propio del dispositivo, traduce el formato binario a un formato más común, como JSON, y envía los datos al sistema previsto sobre IP, por

ejemplo. Si es necesario, la puerta de enlace también puede traducir los comandos enviados desde los sistemas a los dispositivos a tecnologías de comunicación, protocolos y formatos admitidos por el dispositivo respectivo.

2.3.7 Middleware

IoT Integration Middleware (IoTIM) sirve como una capa de integración para diferentes tipos de sensores, actuadores, dispositivos y aplicaciones. Es responsable de (i) recibir datos de los Dispositivos conectados, (ii) procesar los datos recibidos, (iii) proporcionar los datos recibidos a las Aplicaciones conectadas, y (iv) Dispositivos de control. Un ejemplo de procesamiento es evaluar las reglas de acción de condición y enviar comandos a los actuadores en función de esta evaluación. Un dispositivo puede comunicarse directamente con el Middleware de integración de IoT si admite una tecnología de comunicación adecuada, como IP sobre Ethernet o WiFi, un protocolo de transporte correspondiente, como HTTP o MQTT, y un formato de carga compatible, como, por ejemplo, JSON. De lo contrario, el dispositivo se comunica a través de una puerta de enlace con el middleware de integración de IoT. El Middleware de integración de IoT no está limitado a la funcionalidad descrita anteriormente. Puede comprender todo tipo de funcionalidades requeridas por un determinado sistema ciberfísico, por ejemplo, una base de datos de series de tiempo o cuadros de mando gráficos. Además, se puede realizar la gestión de dispositivos y usuarios, así como la agregación y utilización de los datos recibidos. Normalmente, se puede acceder a un middleware de integración de IoT mediante API, por ejemplo, API REST basadas en HTTP.

2.3.8 Aplicación

El componente Aplicación representa un software que utiliza el Middleware de integración de IoT (i) para obtener información sobre el entorno físico y/o (ii) para manipular el mundo físico. Lo hace solicitando datos del Sensor o controlando acciones físicas usando Actuadores. Por ejemplo, un sistema de software que

controla la temperatura de un edificio representa una Aplicación conectada a un Middleware de Integración IoT. Una aplicación también puede ser otro Middleware de integración de IoT.

2.4 PROTOCOLOS IOT

Se utilizan diferentes tecnologías de comunicación entre los dispositivos y la plataforma, dentro de la plataforma y entre la plataforma y los usuarios.

2.4.1. Protocolo de aplicación restringida (CoAP)

El Protocolo de aplicación restringido (CoAP) es un nuevo protocolo de comunicación que está diseñado explícitamente para el hardware de IoT que está inspirado en el Protocolo de transferencia de hipertexto (HTTP) y utiliza la comunicación uno a uno. Dado que CoAP se utiliza para IoT, el hardware debe ser ligero, delgado y generar el menor tráfico posible, y por lo tanto no admite la comunicación TCP/IP. Utiliza el Protocolo de datagramas de usuario (UDP) sobre IP por un lado, y un protocolo más eficiente que el HTTP o la API de transferencia de estado de representación (REST), o similar, por otro lado. Utiliza menos recursos que HTTP e implementa más funciones que HTTP, como observar, ejecutar y descubrir funciones, también, para leer y escribir.[21]

2.4.2. Mqtt

Según el portal mqtt.org MQTT es el acrónimo de Message Queue Telemetry Transport. Es un protocolo de mensajería de publicación/suscripción, extremadamente simple y liviano, diseñado para dispositivos restringidos y redes de bajo ancho de banda, alta latencia o poco confiables. Los principios de diseño son minimizar el ancho de banda de la red y los requisitos de recursos del dispositivo, al mismo tiempo que intentan garantizar la confiabilidad y cierto grado de seguridad de la entrega. Estos principios también hacen que el protocolo sea ideal para el

emergente "máquina a máquina" (M2M) o "Internet de las cosas" del mundo de los dispositivos conectados, y para las aplicaciones móviles donde el ancho de banda y la energía de la batería son importantes. [22]

2.5 SERVICIOS REST

La Transferencia de Estado Representacional (Representational State Transfer) o REST es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.

REST propone una forma de intercambio de mensajes alejada de las abstracciones que se plantean en otros protocolos como SOAP. La idea es comunicar dos máquinas simplemente basándose en el protocolo HTTP y los métodos que éste tiene definidos. Por lo tanto, REST es un protocolo más ligero (lightweight) y no tan encorsetado como SOAP o CORBA.
Los sistemas que siguen los principios REST se llaman con frecuencia RESTful.

2.5.1 Características de rest

Utilizando los métodos especificados por el protocolo HTTP, REST ofrece la posibilidad de crear, leer, actualizar y borrar. Este conjunto de operaciones se conoce como CRUD (Create, Read, Update and Delete) y conforman las operaciones más básicas que se realizan sobre un objeto disponible.
Características de REST:

- Es un protocolo cliente/servidor sin estado
- Proporciona un conjunto de operaciones bien definidas
- Sintaxis universal
- Utilización de hipermedios

• Independiente de plataforma
• Independiente de lenguaje de programación utilizado
• Basado en estándares conocidos (HTTP)

2.5.2 Métodos http en rest

En REST los objetos disponibles a través de la red son tratados como recursos. Sobre estos recursos se aplican las operaciones definidas e implementadas por HTPP. Estos métodos son:

• GET – Recupera la representación definida de un recurso
• PUT – Actualiza una propiedad que afecta a un recurso
• DELETE – Borra un recurso existente
• HEAD – Obtiene las cabeceras del recurso utilizado
• POST – Transfiere información a un programa en una URL dada

Mediante estos métodos se implementan las operaciones sobre los recursos que posteriormente se publican a través de la red. Estos métodos forman lo que sería una API RESTful y facilita una uniformidad en todas las API basadas en REST.

2.6 MIDDLEWARE OPEN-SOURCE

La actual evolución del IoT, trae como respuesta el desarrollo de varios proyectos de software Open-Source, algunos de ellos financiados por organismos internacionales, otros a nivel de empresas de emprendimiento que buscan que su producto sea reconocido y al final pueda ser adquirido por alguna empresa y hay otros desarrollados por comunidades, todo ellos buscan aportar con su tecnología en este nuevo escenario.

Presentamos a continuación los proyectos de Middleware Open-Source más significativos actualmente y como se puede apreciar, son varios proyectos, presentamos una tabla de clasificación para poder agruparlos.

	Chorevolution	DeviceHive	Dsa	FIWARE	IoTivity	IoT-Ignite	Kaa	Kapua	Konker	LinkSmart
Entorno	Java	Java,Go	Java, Dart,Python	~	C,C++	Java	Java	Java	Java	Java
Administración de la Plataforma IoT	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si
Monitoreo	Si	Si	Si	No	No	Si	Si	Si	Si	Si
Alertas/Registro(log)		Via plugins	No	Si	No	Si	Si		Si	Si
Almacenamiento de datos	Si	Si	Si	Si	No	Si	Si	Si	Si	Si
Análisis en tiempo real	Si	Si	Si	Si	No	Si	Si	Si	Si	Si
Registro y administración de dispositivos	Si	Si	No	Si	Si	Si	Si	Si	Si	Si
Interface	REST	REST API, MQTT	REST API	Si	REST API	REST	REST	REST	REST API	REST API
Protocolo	MQTT, CoAP	WebSockets or MQTT	REST API	MQTT,CoAP	CoAP,BLE, etc	MQTT,XMPP, HTTP	via SDK	MQTT, CoAP	MQTT	MQTT, CoAP
Empresa de Soporte		DeviceHive	DGLogik	Telefónica	Open Connectivity	IoT-Ignite® ARDIC - Turquia	KaaIoT	RedHat,Eurotech,Bosch	KonkerLabs	
Licencia	Apache 2.0	Apache 2.0	Apache 2.0	AGPL3.0, Apache2.0	Apache 2.0	Apache 2.0	Apache 2.0	EPL 1.0	CPAL-1.0	Apache 2.0
Fecha de Inicio	~2015	~2012	2016	~2015	~12/2015	~2016	~2015	6/2016 Eurotech	~2013	2015
Documentación	http://www.chorevolution.eu/bin/view/Documentation/WebHome	https://devicehive.com	https://github.com/IOT-DSA/docs/ wiki	https://fiware-iot-stack.readthedocs.io/en/lat est/	https://www.iotivity.org/documentati on	https://devzone.iot-ignite.com/knowledge-base/	http://kaaproject.github.io/kaa/docs/v0.10.0/Welcome/	https://www.eclipse.org/kapua/	https://konker.atlassian.net/wiki/spaces/DEV/pages/28180518/Guia+de+Uso+da+Plataforma+Konker	https://docs.linksmart.eu/
Código Fuente	https://gitlab.ow2.org/chorevolution	https://github.com/devicehive	https://github.com/IOTA	https://github.com/Fiware	https://github.com/iotivity/iot ivity	https://github.com/iot-ignite	https://github.com/kaaproject/kaa	https://github.com/eclipse/kapua	https://github.com/KonkerLabs	https://code.linksmart.eu/
Seguridad	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si
Committers	25	33	~12	~40	~60	~15	56	17	~10	~10

Tabla 3 Plataformas Middleware Open-Source

	OpenIoT	OpenMTC	OpenRemote	SiteWhere	Sofia2	The Things Network	Thinger.io	ThingsBoard	Webinos	WSo2
Entorno	Java	Python	Java	Java,Pythion, Javascript	Java	Go/Python/ Javascript	C++, Java	Java	Javascript	Java,Javascript
Administración de la Plataforma IoT	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si
Monitoreo	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si
Alertas/Registro(log)		Si	Si	Si	Si	Si	No	Si	No	Si
Almacenamiento de datos	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si
Análisis en tiempo real	Si	Si	No	No(Free)	No	No	Si	Si	Si	Si
Registro y administración de dispositivos	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si
Interface	REST	REST	REST API	REST API	REST	MQTT, REST	REST API	REST	REST API	REST API
Protocolo		MQTT	proprietary	MQTT, AMQP, Stomp,	MQTT	LoRaWAN	MQTT, CoAP	MQTT,CoAP,HTTP	MQTT, CoAP	MQTT, WSO2
Empresa de Soporte	OpenIoT	OpenMTC	OpenRemote Inc	SiteWhere	Indra	The things Industries	Thinger.io	ThingsBoard	Webinos	WSO2
Licencia	LGPL V3.0	Eclipse Public License v1.0 (EPL1.0)	GNU AGPL 3,0	CPAL-1.0	Apache2.0	MIT	MIT	Apache2.0	Apache2.0, MIT	Apache 2.0
Fecha de Inicio	2012	2012	~2013	~2013	2012	~2015	2015	~2016	-	2015
Documentación	https://github.com/OpenIotOrg/openiot/wiki/Documentation	https://www.openmtc.org/doc.html	http://www.openremote.com/community/	http://www.sitewhere.org/	https://sofia2.readthedocs.io/en/latest/	http://thethingsnetwork.org/	http://docs.thinger.io/console/	https://thingsboard.io/docs/	http://docs.webinos.org/	https://docs.wso2.com/display/IoTS310/
Código Fuente	https://github.com/OpenIotOrg	https://github.com/OpenMTC/OpenMTC	https://github.com/openremote	https://github.com/sitewhere	https://github.com/Sofia2	https://github.com/TheThingsNetwork/ttn	https://github.com/thinger-io	https://github.com/thingsboard/thingsboard	https://github.com/webinos	https://github.com/ws o2
Seguridad	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si
Committers	12	6	~8	13	-	20-113	-	~25	16	~80

2.6.1 Chorevolution

CHOReVOLUTION es un proyecto de código abierto y su código fuente está disponible gratuitamente bajo la licencia Apache 2.0. Una distribución CHOReVOLUTION lista para máquina está disponible gratuitamente en el portal AppHub en www.apphub.eu.com o directamente desde el enlace de descarga https://l.ow2.org/cpvm.

CHOReVOLUTION aprovecha un nuevo enfoque de ingeniería de software, llamado choreography of services (coreografía de servicios) que hace posible el desarrollo y la ejecución de aplicaciones distribuidas habilitadas para IoT. Las organizaciones que desarrollen habilidades para implementar el entorno integrado CHOReVOLUTION podrán ofrecer servicios efectivos e innovadores en una amplia gama de sectores.

CHOReVOLUTION usa un nuevo entorno integrado de desarrollo y ejecución (IDRE) para desarrollar aplicaciones habilitadas para IoT. CHOReVOLUTION ayuda a crear valor en sectores como Smart Manufacturing, Smart Mobility, Smart City, Smart Retail, a través de aplicaciones distribuidas que integran cosas y servicios. CHOReVOLUTION aprovecha lo último en investigación financiada por la Unión Europea sobre coreografías de servicios, middleware de ejecución y despliegue en la nube de un consorcio de ocho organizaciones europeas.

Las aplicaciones están orientadas al transporte inteligente y turismo inteligente.[23]

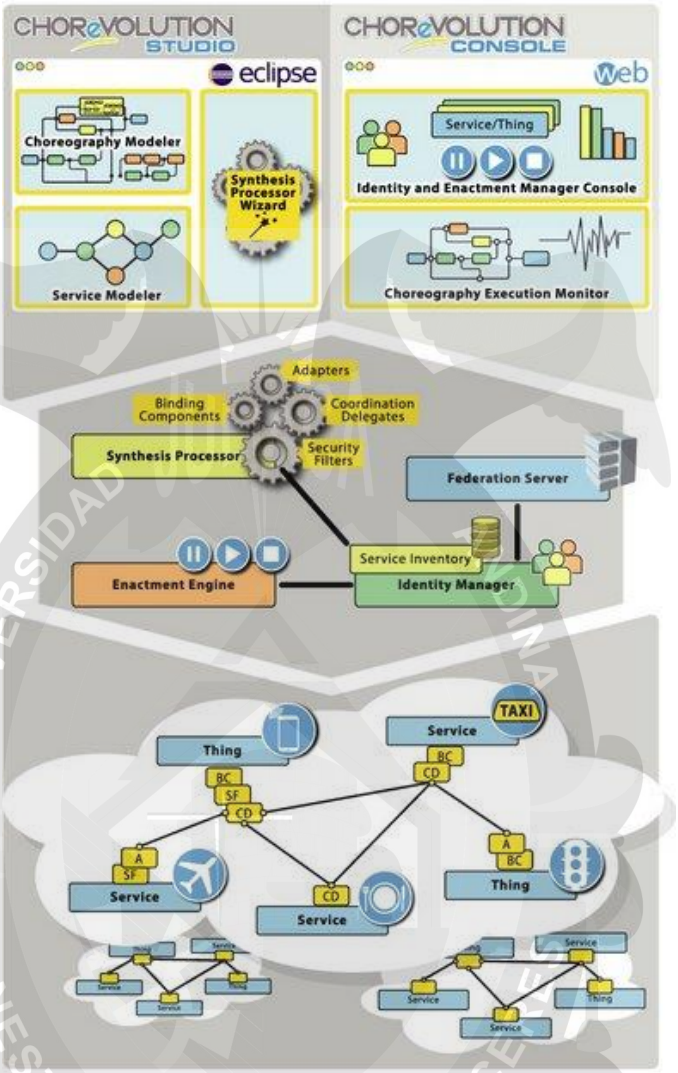


Figura 16 Arquitectura de Chorevolution

2.6.2 Devicehive

Devicehive [24] es otra plataforma OSS IoT que se distribuye bajo la licencia Apache 2.0. Se puede implementar fácilmente a través de Docker y Kubernetes. Se puede conectar a cualquier dispositivo a través de API REST, WebSockets o MQTT.

2.6.3 Dsa

Distributed Services Architecture (DSA) [25] es una plataforma IoT Open Source activa que facilita la intercomunicación, la lógica y las aplicaciones de los dispositivos en todas las capas de la infraestructura de Internet of Things.

2.6.4 Iotivity

IoTivity es un marco de software de código abierto que permite la conectividad perfecta de dispositivo a dispositivo para abordar las necesidades emergentes de Internet of Things.

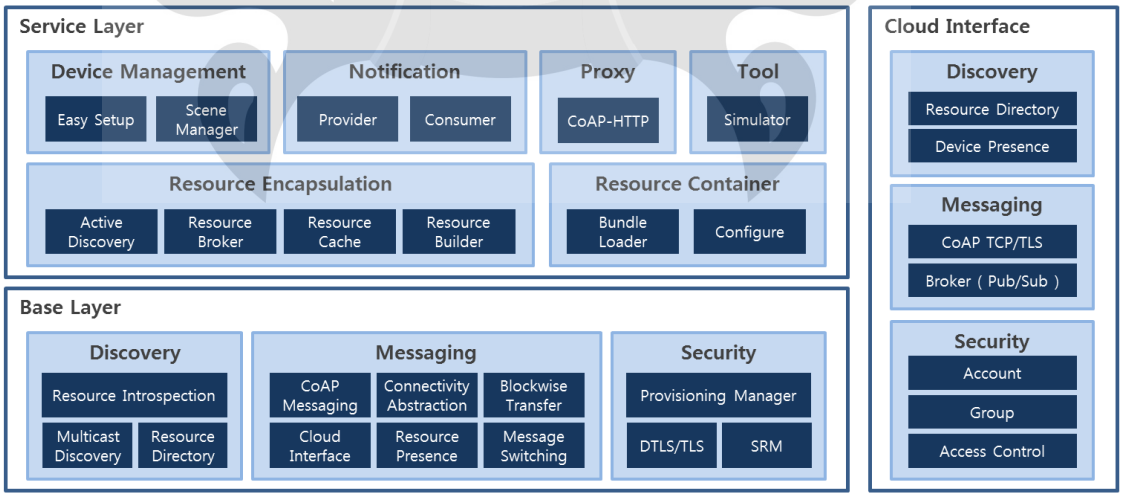
El framework de código abierto de IoTivity es uno de los proyectos patrocinados por el Open Connectivity Foundation (OCF) que se ha desarrollado para proporcionar interoperabilidad entre dispositivos de IoT heterogéneos.

Permite una conectividad de dispositivo a dispositivo (D2D) sin inconvenientes y se dirige a varios dominios de aplicaciones, como la automatización del hogar y la fabricación, la atención médica y las redes sociales.

IoTivity utiliza, sin embargo, mucha memoria en pequeños dispositivos que tienen recursos de hardware limitados y ejecuta el Sistema Operativo en Tiempo Real (RTOS).[26]

Figura 17 Arquitectura de Framework IoTivity

- **IoTivity v1.2**



2.6.5 Fiware

Fiware es una arquitectura que permite construir un ecosistema abierto y sostenible en torno a estándares de plataforma de software públicos, libres de derechos y basados en la implementación que facilitarán el desarrollo de nuevas aplicaciones inteligentes en múltiples sectores [27] . Fiware maneja el concepto de GE (Generic Enabler) que es una librería de servicios de propósito general que cubren funcionalidades comunes en campos como seguridad, almacenamiento, cloud, data context e Internet of Things. Estos servicios están disponibles a través de API'S disponibles para que los desarrolladores las puedan adoptar en la construcción de aplicaciones.

Arquitectura de Fiware

La arquitectura de Fiware por si sola es muy extensa y vas más allá del objetivo de este trabajo de fin de máster por lo que solo se hará énfasis en la arquitectura de IoT que la propia Fiware proporciona para motivos académicos [28]. En la figura 18 se puede observar esta arquitectura donde tenemos el IoT Device Management (IDAS), el Orion Context Broker, el IoT Gateway y las aplicaciones. El Gateway Logic GE es un componente opcional que proporciona funciones de gestión de la IoT Edge API y actúa como puerta de enlace entre el API y las funciones.

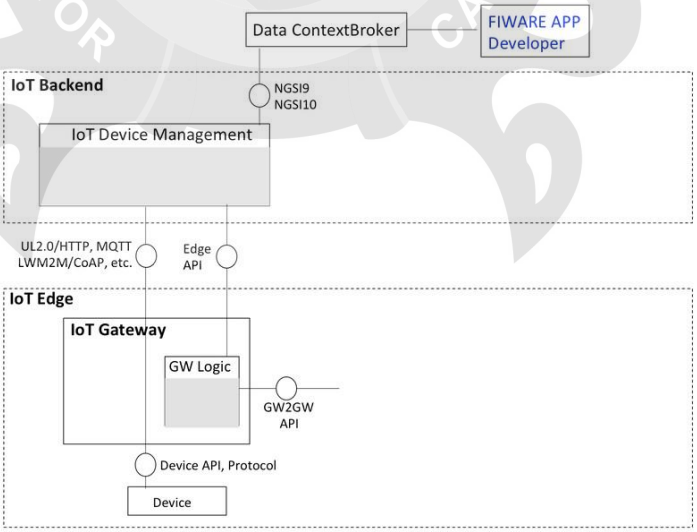


Figura 18 Arquitectura IoT de Fiware.

2.6.6 IoT-Ignite

IoT-Ignite es una plataforma como servicio (PaaS) diseñada específicamente para la realización de Internet of Things. Detrás del diseño, hay mejores prácticas modernas y una arquitectura única. Los Servicios de IoT existentes envían datos directamente desde dispositivos periféricos a la nube o pasan datos de fila a través de puertas de enlace sin procesamiento. En ambos casos, el tráfico de datos de gran volumen se transporta entre el borde y la nube. IoT 2.0 permite una capacidad de transferencia de datos significativa al procesar los datos dentro de las puertas de enlace.[29]

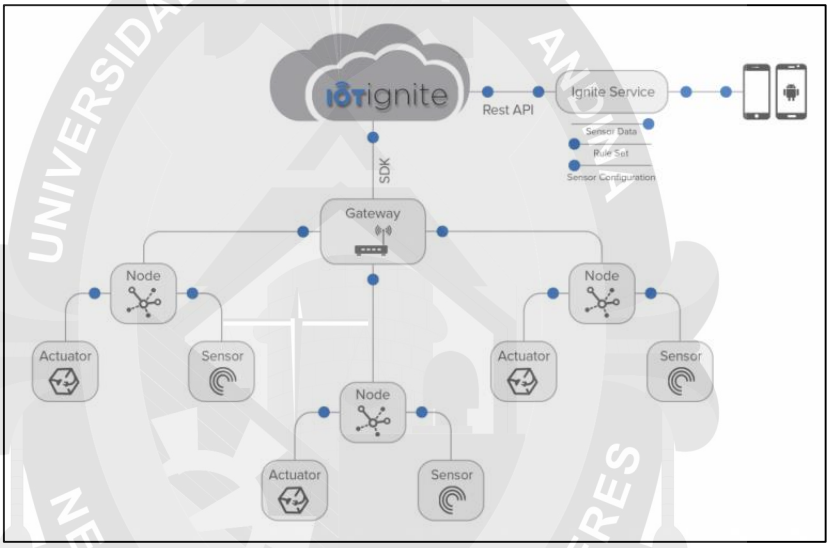


Figura 19 Arquitectura IoT-Ignite

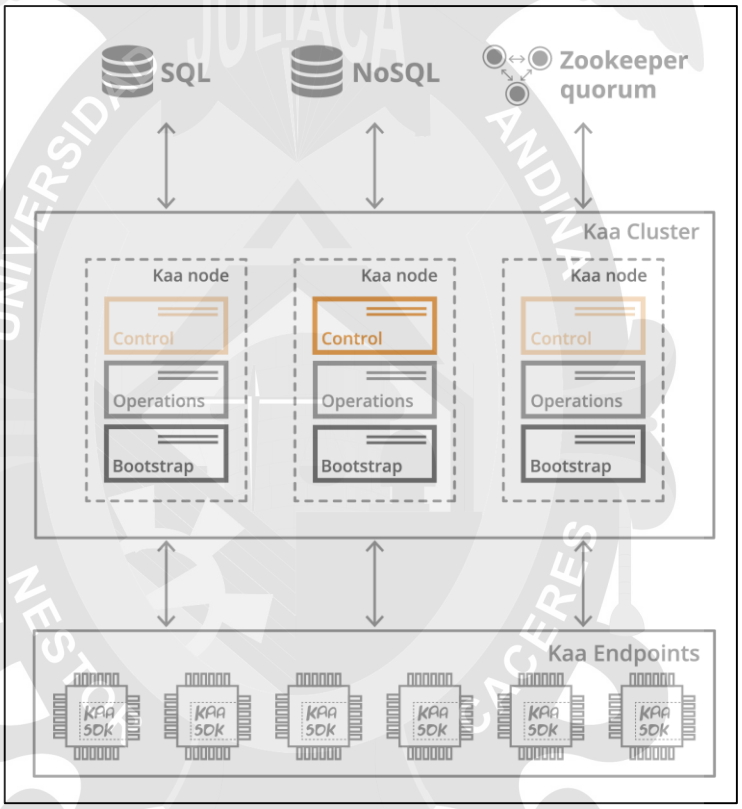
Fuente [30]

2.6.7 Kaa

Kaa es una plataforma de middleware de código abierto creada y mantenida por KaaIoT y licenciada bajo la licencia Apache 2.0. A pesar de que es un código abierto, los usuarios pueden expandirse a una versión paga contactando con KaaIoT. Admite las comunicaciones REST con su servidor, y los SDK se pueden implementar en los dispositivos. Para implementar con éxito la solución, los usuarios deben tener Oracle Java SDK, ya sea MariaDB o PostgreSQL, MongoDB o Cassandra, y Zookeeper. La desventaja de Kaa (al implementar un servidor privado)

es que no es posible consultar los datos almacenados del servidor a través de la API REST, lo que significa que el usuario tiene que desarrollar otra aplicación para esta característica. Para aquellos que estén interesados, es posible construir una API REST que devuelva datos de una base de datos MongoDB usando herramientas gratuitas como Spring tool suite. Se puede encontrar más información sobre Kaa en su sitio web oficial [31].

Figura 20 Arquitectura de Kaa



Fuente: [32]

2.6.8 Kapua

Kapua [33] es una plataforma para gestionar las puertas de enlace IoT respaldadas por la fundación Eclipse. Kapua ofrece gestión de dispositivos, datos y registro, así como servicios de mensajería. Permite la administración remota y la integración fácil con otras aplicaciones a través de la API RESTful. También incluye almacenamiento

de datos para análisis, así como tableros para visualizaciones. Sin embargo, parece que no es bastante genérico, sino que está más orientado a soluciones específicas integradas con Eclipse KURA. El código fuente está disponible en [34], se mantiene activamente con las empresas principales de respaldo Red Hat y Eurotech.

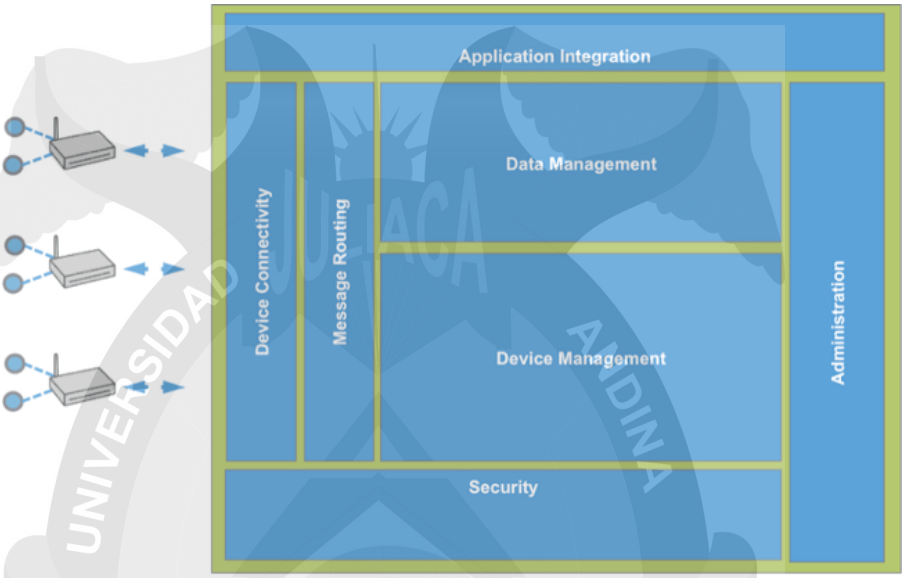


Figura 21 Arquitectura Eclipse Kapua

Fuente [35]

2.6.9 Konker

Konker es un middleware de código abierto creada y mantenida por los laboratorios brasileños KonkerLabs. Está licenciado bajo la licencia de Apache 2.0. A pesar de que es una fuente abierta, una versión en línea está disponible como PaaS, donde los usuarios pueden probar de forma gratuita o ampliar a una versión de pago. Admite las comunicaciones REST y MQTT con su servidor. Para implementar con éxito la solución, los usuarios deben tener Java SDK, MongoDB, Cassandra, un servidor de aplicaciones que admite servlets. Se puede encontrar más información sobre Konker en su sitio web oficial [36].

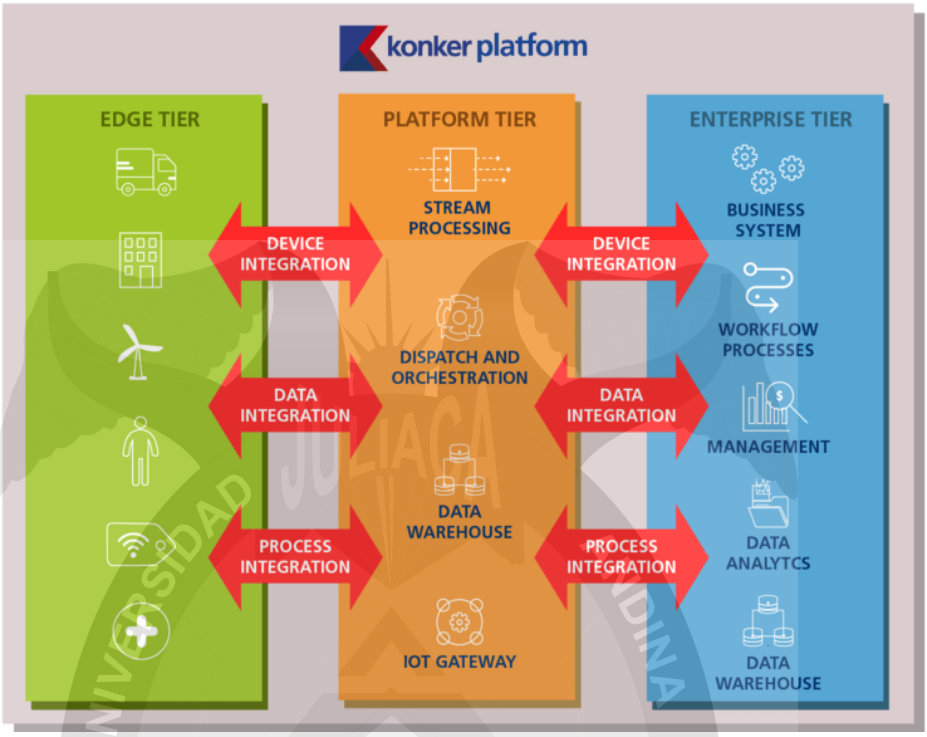


Figura 22 Arquitectura de Konker

2.6.10 LinkSmart

Antes conocido como Hydra, es una plataforma completa de IoT que admite la gestión de dispositivos, así como la habilitación de aplicaciones. El módulo de habilitación de la aplicación se llama Linksmart HDS (Historical Datastore). HDS es una plataforma de middleware de código abierto que tiene licencia de Apache license 2.0. Admite las comunicaciones REST con su servidor, y la visualización de datos se realiza a través de grafana. Para implementar con éxito la solución, los usuarios deben tener instalado influxDB o MongoDB. Con respecto a las plataformas que se experimentaron, es el único que usa SenML en lugar de JSON. Se puede encontrar más información sobre Linksmart en su documentación oficial [37].

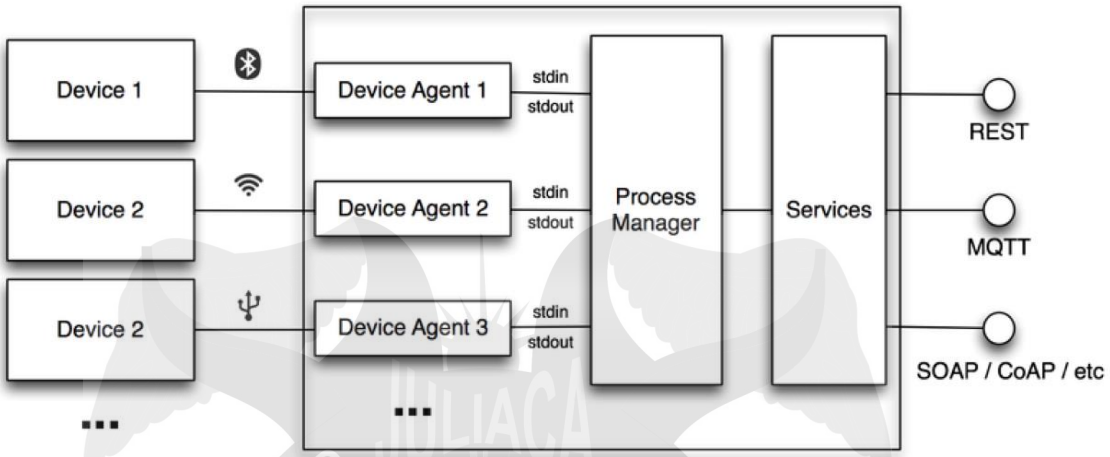


Figura 23 Arquitectura de LinkSmart

Fuente: [38]

2.6.11 OpenIoT

Es una plataforma de código abierto que admite la gestión de dispositivos y la habilitación de aplicaciones. Creado y mantenido por el consorcio OpenIoT, El proyecto OpenIoT se lanza bajo la licencia LGPL V3.0. Admite las comunicaciones REST y GSN (Global Sensor Network) con su servidor. Para implementar con éxito la solución, los usuarios deben tener instalados Java, Maven, JBoss y Local Virtuoso. Aunque es un proyecto fascinante, no ha recibido actualizaciones de su repositorio de Github desde agosto de 2017. [39]

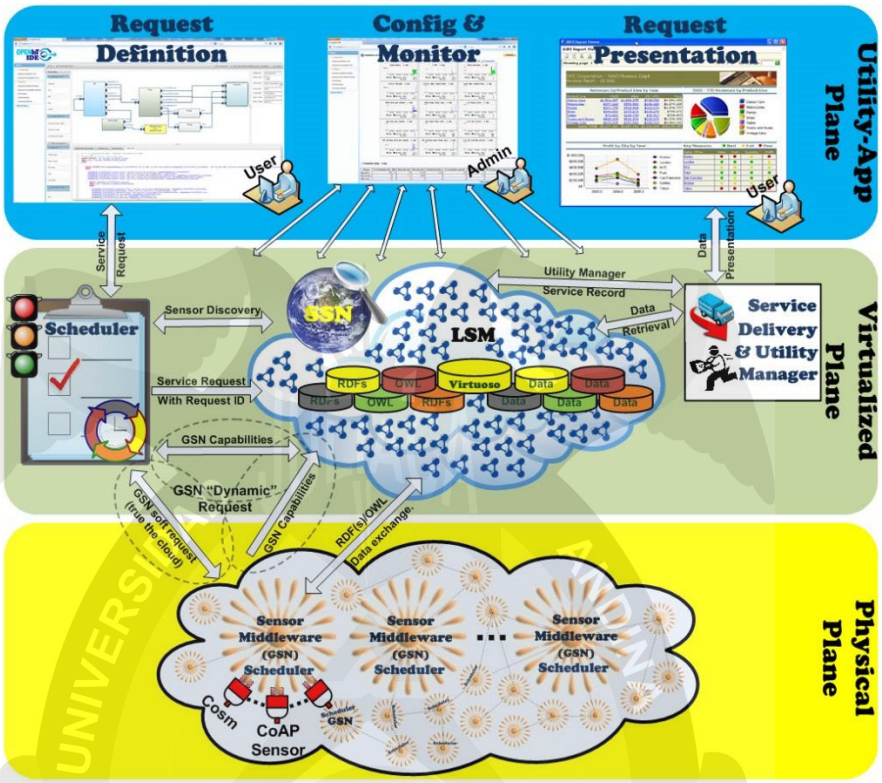


Figura 24 Arquitectura OpenIoT

Fuente [40]

2.6.12 OpenMTC

Es una plataforma de IoT open-source y habilitada para la nube, y su comparación con nuestra arquitectura de referencia IoT. OpenMTC se compone de los siguientes componentes: Front y Back-End, el componente de sensores y actuadores debajo del front-end, la conectividad entre el front-end y el back-end, las aplicaciones ubicadas en la parte superior del back-end y en el el lado derecho del front-end, así como un componente para conectar otras plataformas M2M al back-end. La documentación de OpenMTC describe además que los componentes de sensores y actuadores junto con el nivel más bajo del front-end, que permite la comunicación, representan dispositivos equivalentes a nuestra arquitectura de referencia. Las características principales y los componentes de conectividad del Front-End, la conectividad entre el Front-End y Back-End, así como el componente de

conectividad del back-end proporcionan todas las funciones necesarias para habilitar la comunicación entre un dispositivo y el middleware, tales como como traducción de mensaje. En consecuencia, esos componentes representan nuestra puerta de enlace. Dado que el componente OpenEPC (que ofrece conectividad entre el front-end y el back-end) proporciona una funcionalidad adicional, como la aplicación de reglas y el filtrado, también lo abarca el Middleware de integración de IoT. Además, IoT Integration Middleware encapsula los componentes de conectividad, funciones principales y habilitación de aplicaciones del componente de fondo de OpenMTC. Esos componentes proporcionan la funcionalidad principal de la plataforma. Los componentes de Application Enablement proporcionan toda la funcionalidad para conectar otras aplicaciones al middleware. [41]

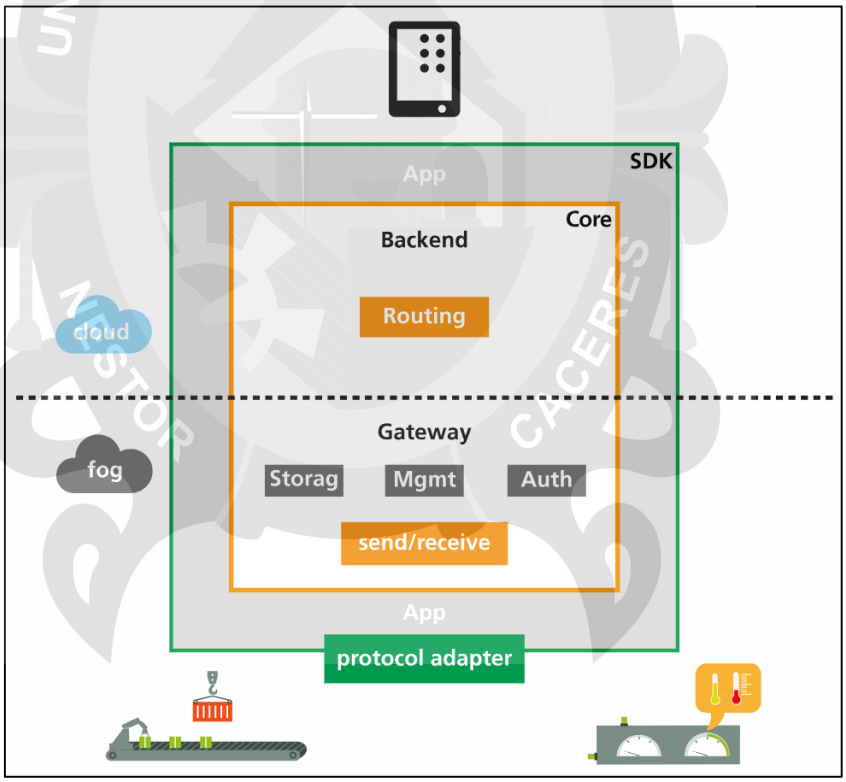


Figura 25 Arquitectura OpenMTC

2.6.13 OpenRemote

OpenRemote [42] es un proyecto de código abierto que ofrece tres elementos principales alrededor de herramientas de configuración de nube (OpenRemote Manager), un controlador para integrar protocolos que actúan como puerta de enlace (OpenRemote Controller) y reglas de administración y panel de control de UI para diseñadores (OpenRemote Designer). OpenRemote Manager es la parte de la nube IoT de toda la solución y ofrece API para servicios de terceros. Es mantenido activamente por la compañía Openremote Inc y el código fuente está disponible en [43].

2.6.14 SiteWhere

SiteWhere es una plataforma de IoT de fuente abierta. Proporciona un sistema que facilita la ingestión, el almacenamiento, el procesamiento y la integración de los datos del dispositivo. SiteWhere proporciona la siguiente funcionalidad:

- Plataforma IoT Server
- Administración del Dispositivo
- Integración[44]

Arquitectura de SiteWhere

Componentes globales

Como se muestra en el diagrama de la arquitectura, SiteWhere se compone de muchos componentes diferentes que están conectados entre sí para proporcionar la plataforma central. En las secciones a continuación, cubriremos los componentes que son globales para el sistema. Todos los inquilinos comparten estos ajustes globales.

Contenedor de aplicaciones web

SiteWhere se implementa como un archivo de archivo de aplicaciones web (WAR) y está diseñado para ejecutarse en un contenedor web como Apache Tomcat. SiteWhere se ejecutará en una versión estándar de Apache Tomcat suponiendo que los archivos de configuración se copian en la carpeta conf de Tomcat . Los archivos de configuración pueden modificarse para cambiar la forma en que SiteWhere procesa los eventos del dispositivo y se integra con los servicios externos.

SiteWhere Server

SiteWhere Server es la aplicación central que controla todos los demás componentes de SiteWhere.

Aplicación administrativa

SiteWhere incluye una aplicación administrativa HTML5 que se puede usar para administrar cómo funciona el sistema.

Servicios REST

La mayoría de la funcionalidad principal relacionada con las API de SiteWhere se puede acceder de forma externa a través de los servicios REST.

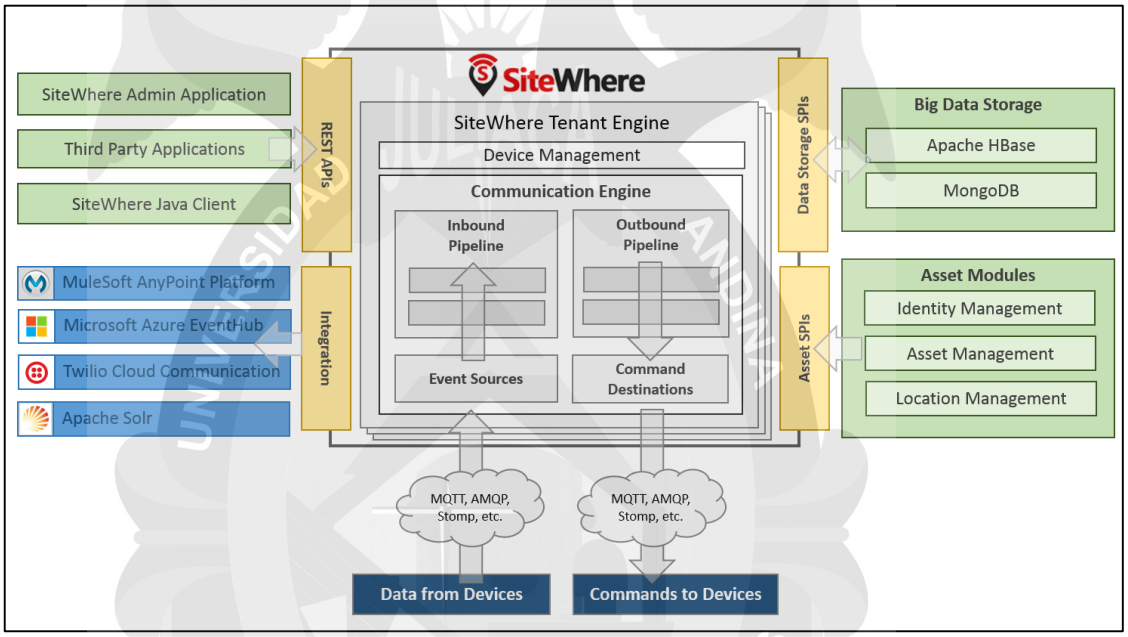
Datastore global

Al almacenar y recuperar datos, SiteWhere nunca trata directamente con una base de datos. En cambio, el sistema define las Interfaces de Proveedor de Servicio (SPI) para las operaciones de datos que necesita para operar y espera que las implementaciones del almacén de datos cumplan con las interfaces requeridas. El almacén de datos de administración de usuarios se configura a nivel global y se basa en las siguientes API:

IUserManagement : contiene todas las llamadas de administración central de usuarios, incluidos los métodos CRUD para usuarios, autoridades, etc.

Al configurar una nueva instancia de servidor de SiteWhere, cambia la configuración en el archivo de configuración principal de Spring para indicar qué tipo de datastore usar para la implementación de datos subyacente. Los tipos de datastores actualmente admitidos incluyen MongoDB y Apache HBase. [45]

Figura 26 Arquitectura SiteWhere



Fuente: [45]

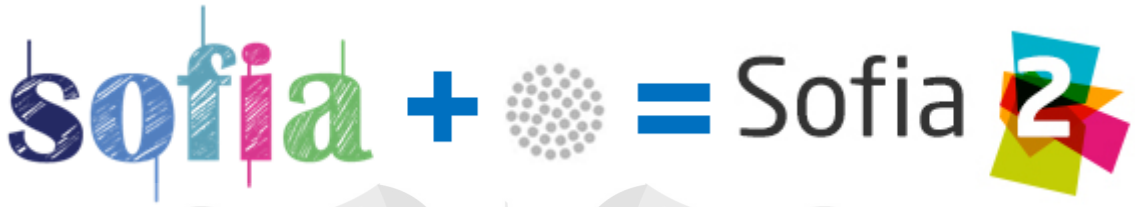
2.6.15 Sofia2

Sofia2 surge de un proyecto I+D europeo denominado SOFIA.

SOFIA es el acrónimo de SMART OBJECTS FOR INTELLIGENT APPLICATIONS y es una plataforma que surge de un proyecto de I +D Artemis de tres años finalizado en Marzo de 2012, en el que participan 19 partners de cuatro países de la UE, entre los cuales están Nokia, Philips, Fiat, Acciona e Indra.

Tras el proyecto Artemis Indra ha seguido evolucionando el proyecto SOFIA original creando una plataforma enfocada a su uso empresarial: Sofia2 [46]

Figura 27 Sofia2



Fuente: [46]

En [47] se describe a Sofia2 como la unión de un middleware y repositorio capaz de procesar miles de eventos por segundo, con almacenamiento en tiempo real y Big Data, con análisis, modelado visual y reglas integradas, multiplataforma con interfaces multiprotocolo y todo operable desde una Consola Web. Todo esto permite que diferentes dispositivos IoT heterogéneos puedan convivir en un mismo ecosistema a través del SDK y las API's que la plataforma proporciona. Según [48] Sofia2 está centrada en ofrecer servicios de IoT en la nube, donde pueden implementarse aplicaciones.

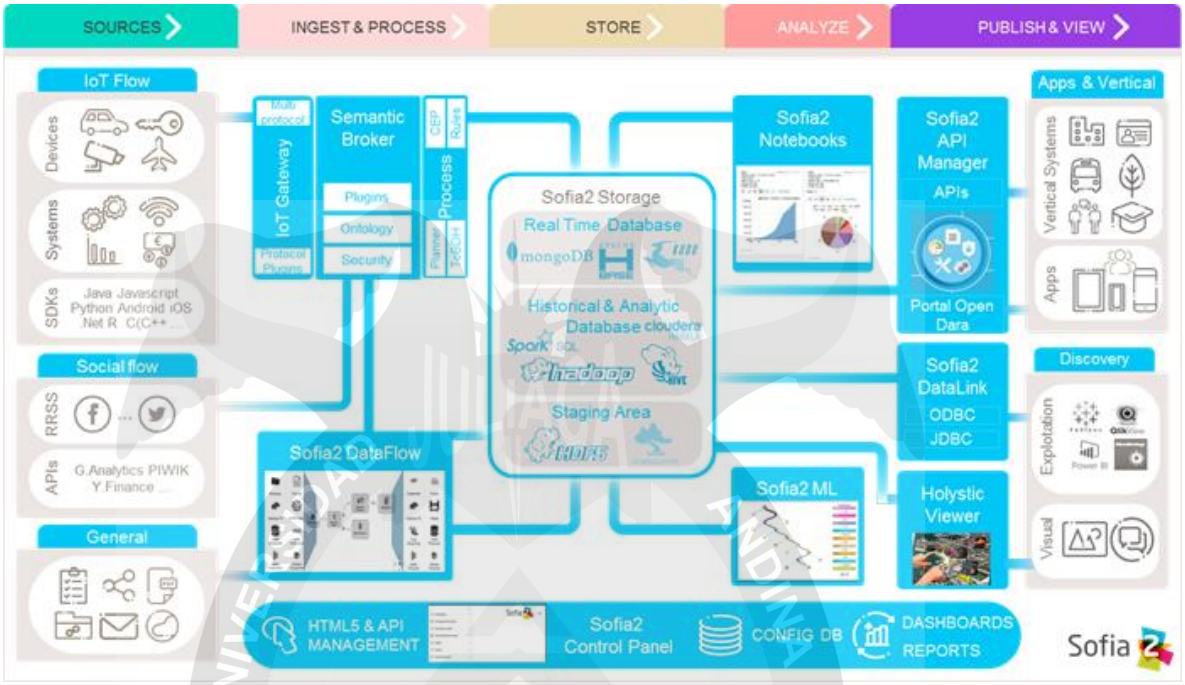
Arquitectura de Sofia2

En la figura 28 se describe la Plataforma desde el punto de Vista de los módulos funcionales que componen la Plataforma.

Éstos son los módulos necesarios para dar soporte a sistemas IoT:

- SDKs
- Sofia2 Control Panel
- IoT Gateway
- Semantic Broker
- Process
- Sofia2 Storage
- Sofia2 API Manager
- Holystic Viewer

Figura 28 Arquitectura Global Sofia2



Fuente: [49]

2.6.16 The things network

The Things Network [50] se creó en 2015 y está basado en la tecnología LoRaWAN. Es una red de IoT abierta y gratuita en todo el mundo que consta de cientos de puertas de enlace. Se incluye en este informe ya que The Things Network (TTN) se ofrece como Open Source, se encuentra en [51] y se puede usar para una implementación privada o para aportar recursos a la nube TTN IoT. La plataforma TTN IoT Cloud ofrece la mayoría de los servicios previstos de una capa de IoT en la nube: conectividad con gateways LoRaWAN de extremo, gestión de dispositivos y MQTT, así como una API RESTful para aplicaciones de usuario final.

2.6.17 Thinger

Thinger.io [52] es una plataforma OSS que asegura una fácil escalabilidad. Ofrece consolas de administración y API REST para la integración de aplicaciones y una

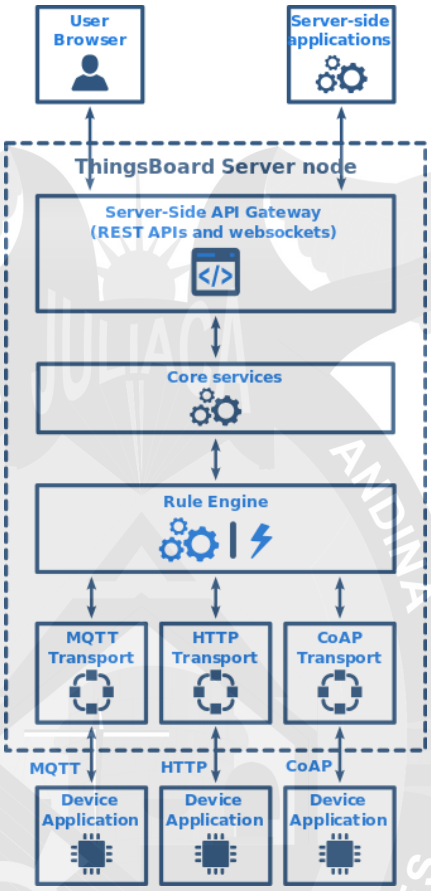
fácil integración con Docker. Ofrece también bibliotecas para Arduino y Android. El código fuente se encuentre en [53].

2.6.18 Thingsboard

Es una plataforma de IoT open-source disponible desde un repositorio de GitHub (https://github.com/thingsboard/thingsboard) bajo la licencia de Apache versión 2.0. La compañía que está detrás de la plataforma también ofrece una edición profesional comercial con soporte adicional e integraciones adicionales de la plataforma. La arquitectura general de ThingsBoard se muestra en la figura 30. La conectividad con dispositivos se maneja a través de diferentes componentes de transporte. Además de la plataforma IoT, también está la puerta de enlace ThingsBoard IoT para integrar dispositivos IoT conectados a sistemas de terceros con ThingsBoard.

Los datos se pueden almacenar en una base de datos PostgreSQL o Cassandra externa. ThingsBoard utiliza Apache Zookeeper para la coordinación del clúster y Cassandra como una base de datos NoSQL. Los servicios principales son responsables de la administración del dispositivo, la administración del usuario y los paneles. La pasarela API del lado del servidor proporciona una puerta de enlace REST que permite el acceso a datos de series de tiempo, y también permite a los usuarios registrados enviar comandos a los dispositivos. ThingsBoard tiene una arquitectura plug-in que permite el acoplamiento a componentes externos. Los plug-ins existentes para Apache Kafka y el envío de correos electrónicos están disponibles. Internamente, ThingsBoard usa Akka para el procesamiento de mensajes impulsado por eventos.[54]

Figura 29 Arquitectura Thingsboard



Fuente: [55]

2.6.19 Webinos

Es un middleware de código abierto para IoT y dispositivos móviles, patrocinado por el proyecto FP7 de la Unión Europea. El objetivo de Webinos es proporcionar un marco seguro para que los dispositivos personales se comuniquen y para que las personas publiquen datos a terceros y a otras personas. Como tal, toma un enfoque diferente a una plataforma de IoT al estar centrado alrededor de la persona. El mapeo del enfoque de Webinos en nuestra arquitectura de referencia IoT se muestra en la figura 31. Los componentes principales de la arquitectura de Webinos son el Hub de Zona Personal (PZH) y el Proxy de Zona Personal (PZP). El PZH proporciona la puerta de enlace, donde se conecta cada dispositivo. El PZH también

proporciona comunicaciones locales entre dispositivos actuando como un centro de mensajería. En este sentido, realiza las funciones de nuestro Middleware de integración de IoT. El PZH no admite inherentemente ninguna aplicación para ejecutarse localmente, pero proporciona las API que permiten construir aplicaciones de terceros y comunicarse con dispositivos, que es una función central de la capa de middleware de integración de IoT en nuestra arquitectura. Cada dispositivo ejecuta un componente local, el PZP, que agrega los datos del sensor y los comandos del actuador y se comunica con el PZH. Un aspecto único de Webinos es que cuando varios PZP (en varios dispositivos) se han conectado al mismo PZH, pueden comunicarse de igual a igual. El PZP y PZH se sincronizan para permitir que esto suceda.[56]

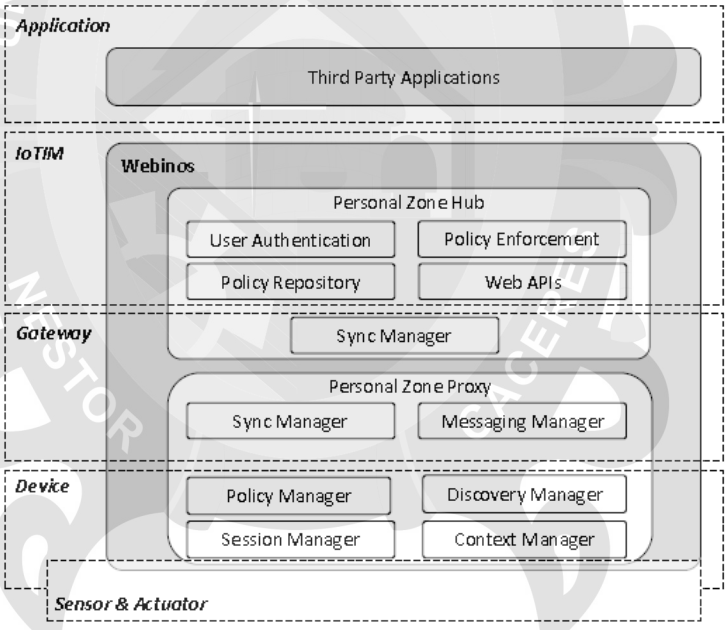


Figura 30 Arquitectura Webinos

2.6.20 Wso2

El software de código abierto de WSO2[57] le ofrece agilidad y flexibilidad para recomponer el código o realizar cambios y adiciones para cumplir con sus requisitos específicos a medida que evolucionan. Permite a las empresas desarrollar su trabajo, fomentando la agilidad del equipo y la integración superior de las aplicaciones. Creemos que el código abierto es el futuro de la industria.

WSO2 Identity Server es completamente de código abierto y se libera bajo Apache Software License Versión 2.0, una de las licencias más amigables para empresas disponibles en la actualidad. WSO2 Identity Server es un proyecto en curso. Se somete a mejoras y mejoras continuas con cada nueva versión, para abordar los nuevos desafíos comerciales y las expectativas de los clientes.

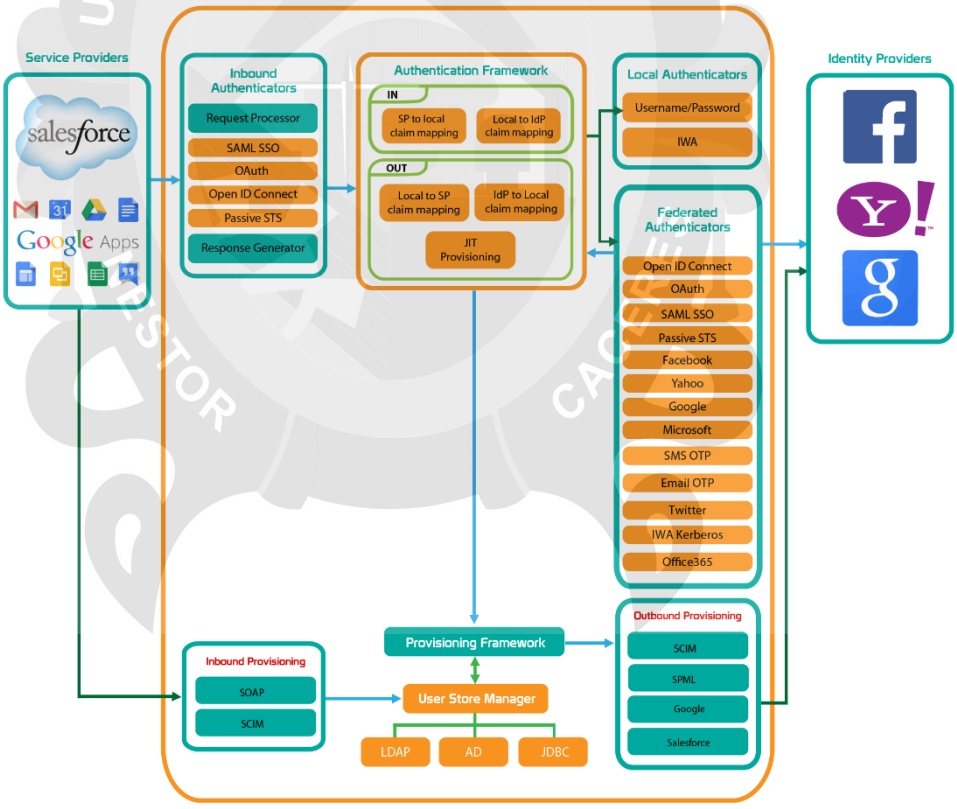


Figura 31 Arquitectura WSo2

Fuente [58]

2.7. METRICAS DE RENDIMIENTO PARA EL MIDDLEWARE IOT

2.8 ÍNDICE DE MEDICIÓN DEL SERVICIO

De sus siglas en inglés Service Measurement Index (SMI), fue iniciado en el año 2010, por dos científicos de la Universidad Carnegie Mellon Silicon Valley, Jane Siegel y Jeff Perdue, lanzaron una iniciativa, que textualmente indica:
"Estamos ayudando a desarrollar un conjunto de medidas centradas en el negocio, combinando datos cuantitativos y cualitativos que proporcionarán a los directores de información un método estandarizado para comparar servicios en la nube de proveedores internos o externos".[59]

Se formó un consorcio liderado por la Universidad Carnegie Mellon Silicon Valley denominado "Consorcio de Índices de Medición de Servicios en la Nube" (CSMIC).
El Índice de medición del servicio (SMI) es un conjunto de indicadores clave de rendimiento, Key Performance Indicator (KPI, por sus siglas en inglés) relevantes para la empresa que proporcionan un método estandarizado para medir y comparar un servicio comercial, independientemente de si el servicio se proporciona internamente o proviene de una compañía externa.
Está diseñado para convertirse en un método estándar para ayudar a las organizaciones a medir los servicios empresariales basados en la nube en función de sus requisitos empresariales y tecnológicos específicos.
El Índice de medición de servicios está siendo desarrollado actualmente por el Cloud Service Measurement Index Consortium (CSMIC).
El SMI es un marco jerárquico. El nivel superior divide el espacio de medición en 7 categorías. Cada categoría se refina aún más por 4 o más atributos.
Luego, dentro de cada atributo, se define un conjunto de KPI que describen los datos que se recopilarán para cada medida o métrica. Algunos de estos KPI serán específicos del servicio, mientras que otros se aplicarán a todos los servicios de la nube como IaaS, PaaS y SaaS.

Figura 32 Las 7 categorías del Índice de Medición del Servicio - SMI

Fuente: http://csmic.org/downloads/SMI_Overview_TwoPointOne.pdf

El Índice de medición del servicio (SMI) [60] define siete categorías principales que deben considerarse al comparar las plataformas middleware IoT, que son una combinación de medidas cuantitativas y cualitativas. Estos son:

- Responsabilidad.
- Agilidad.
- Garantía.
- Financiero.
- Desempeño ó rendimiento,
- Seguridad y Privacidad.
- Usabilidad.

Cada categoría tiene múltiples atributos, que son de naturaleza cuantitativa o cualitativa. Por ejemplo, los atributos cuantitativos, como el tiempo de respuesta del servicio, la precisión, la disponibilidad y el costo, pueden medirse cuantitativamente

mediante el uso de herramientas de monitoreo de software, mientras que los atributos cualitativos como la facilidad de uso, la flexibilidad, la idoneidad, la operabilidad, la elasticidad, etc., no pueden cuantificarse. En su mayoría se deducen en base a las experiencias del usuario. Estos atributos cualitativos se miden utilizando una escala ordinal que consiste en un conjunto de calificaciones predefinidas, como buena, alta, media, regular, excelente, etc.

Presentamos a continuación una tabla en donde clasificamos las métricas según la categoría del índice de medición del servicio (SMI) en dos grupos principales cuantitativos y cualitativos, en el caso de las métricas cuantitativas, latencia, bytes enviados, bytes recibidos, tiempo de conexión, se acompaña la unidad de medida que será evaluada por el software Apache JMeter.

Para la métrica de disponibilidad estará dada por el servidor en la nube, información proporcionada por el proveedor que generalmente es un valor cercano al 100% esto en las clasificaciones de datacenters.

Referente al costo este nos indicará el valor monetario de la plataforma IoT, el trabajo de investigación solo considera los middleware open-source, por consiguiente, el costo será cero (US$ 0.00).

Tabla 4 Clasificación de las Metricas Cuantitativas y Cualitativas según SMI

Tipo de Métrica	Métrica	Categoría SMI	Definición	Unidad de Medida
Cuantitativa	Latencia	Performance (Desempeño)	Medida del tiempo entre el momento en que se realiza una solicitud y se emite una respuesta para el usuario.	ms
	Bytes Enviados y Bytes Recibidos	Performance (Desempeño)	Cantidad de bytes enviados y Recibidos por cada sensor y numero de parámetros.	bytes
	Tiempo de Conexión	Performance (Desempeño)	Tiempo de duración en milisegundos de la conexión establecida entre el sensor y el Middleware IoT.	ms
	Disponibilidad	Assurance (garantia)	Medida de la probabilidad de la duración del tiempo en que el servicio estará en funcionamiento sin tiempo de inactividad	%
	Costo	Financial (Financiera)	Costo de adquisición o costo de uso de un servicio por parte de un usuario	US$
Cualitativa	Usabilidad	Usability (Usabilidad)	Facilidad con la que el usuario puede utilizar.	Alto Medio Bajo
	Instalación	Usability (Usabilidad)	Calificación de la complejidad de instalar el middleware.	fácil mediana compleja
	Aprendizaje	Usability (Usabilidad)	Facilidad de aprender a usar el Middleware.	Alto Medio Bajo
	Seguridad	Security and Privacy (Seguridad y Privacidad)	Medida en que un servicio puede satisfacer los requisitos de seguridad del usuario en términos de control de acceso, privacidad, datos, infraestructura, etc.	Alto Medio Bajo
	Flexibilidad	Agility (Agilidad)	Calificación de la capacidad de agregar o eliminar funciones predefinidas de un servicio para adaptarse a las preferencias de los usuarios.	Alto Medio Bajo

Fuente: Elaboración propia.

2.9 MÉTRICAS CUALITATIVAS

Las métricas cualitativas son métricas que son difíciles de traducir en números porque la forma en que se perciben depende de la persona que está analizando. Hay muchas plataformas de middleware de IoT disponibles; por lo tanto, las métricas cualitativas se usan para filtrar el middleware.

Las plataformas de middleware IoT son tan similares en características que la comparación cualitativa tradicional donde una tabla que contiene una lista de características (que refleja sus requisitos o detalles de aspectos arquitectónicos), no ayuda a decidir qué middleware usar. Imaginemos, por ejemplo, que una solución está marcada como no escalable, el razonamiento detrás de dicha declaración debe detallarse al máximo. Las métricas cualitativas en este trabajo se proponen con el objetivo de reducir la subjetividad, así como también ayudar a los usuarios que intentan seleccionar un middleware de IoT para una solución determinada.

2.9.1 Usabilidad

El termino usabilidad ha sido definido por varias normas de Organizaciones Internacionales de Estándares de Calidad como es el ISO, y IEEE. En cada norma la usabilidad está relacionada a la calidad del mismo.

Se reconoce a la usabilidad como "la capacidad en que un producto de software puede ser entendido, aprendido y usado por determinados usuarios bajo ciertas condiciones en un contexto de uso específico". Se contempla la calidad interna, externa y en uso de un producto de software. A su vez, la usabilidad es descompuesta en subatributos, haciendo que algunos atributos sean más tangibles y se puedan medir [61].

La norma ISO 25000 (Square) [62] contempla a la usabilidad bajo dos puntos de vista distintos: uno que contempla a la usabilidad desde el punto de vista del software, como producto en sí mismo; y el otro punto de vista desde la usabilidad de uso, desde la perspectiva del usuario.

A través de los distintos estándares se definen distintos atributos de la usabilidad, que sirven para formular métricas para la evaluación del software.[63]

2.9.2 Instalación

Es el procedimiento que se debe seguir para desplegar la plataforma middleware, este proceso en la medida de las posibilidades debe estar documentado, y fácil acceso en algún repositorio público de software, como github, indicando cuáles son sus características técnicas, versión del middleware, sistema operativo recomendado, procesador, memoria mínima requerida, espacio en disco duro. Si se dispone de algún contenedor como docker, o quizás existan una versión de máquina virtual que se pueda ejecutar en forma local.

Esta valoración cualitativa la podemos clasificar en fácil, mediana y compleja, el usuario es el encargado de dar la valoración.

2.9.3 Aprendizaje

La capacidad de aprendizaje se define en CSMIC SMI como "El esfuerzo requerido para que los usuarios aprendan a usar el servicio". Para que los servicios basados en la nube sean igual de atractivos o más atractivos para los clientes que los servicios que no son de la nube, el esfuerzo por aprender a usar el servicio debe tener un umbral bajo y una cantidad mínima de tiempo y experiencia asociados con estar listos para usar el servicio.

Los middlewares estudiados, Konker y ThingsBoard, han requerido un tiempo para poder aprender a usar la plataforma, podemos indicar en cuanto a su valoración de aprendizaje que la plataforma Konker es baja, mientras que la plataforma ThingsBoard la podemos valorar como media, otro factor que pude influir en el aprendizaje, como es el caso de Konker la interfaz de usuario está en el idioma portugués y la de ThingsBoard, tiene una interfaz en el idioma español.

2.9.4 Protocolos de aplicación compatibles

Para maximizar la compatibilidad con los dispositivos, la plataforma middleware debe admitir una gran cantidad de protocolos de aplicación, ya que garantiza que una amplia gama de productos sea compatible con la plataforma middleware. Sin embargo, el obligatorio debe ser HTTP(S) (Protocolo de transferencia de hipertexto), ya que la capa de la aplicación de Internet se basa en él. Otros protocolos estándar de IoT son MQTT (Message Queue Telemetry Transport) y CoAP (protocolo de aplicación restringida) [64] . Aquí, los protocolos soportados se refieren a los protocolos que se pueden usar para comunicarse con la plataforma middleware IoT (es decir, capa de aplicación).

2.10 MÉTRICAS CUANTITATIVAS

Las medidas cuantitativas se convierten fácilmente en números y se pueden cuantificar.

Una comparación justa entre diferentes soluciones implica que las condiciones son las mismas (para todas las soluciones) en todos los aspectos considerados. Para el software, esto significa que la máquina host tendrá los mismos recursos disponibles (memoria, potencia de procesamiento, espacio en disco, etc.). En la práctica, esto significa que, con más recursos, la instancia local puede funcionar mejor en comparación con menos recursos. Las comparaciones cuantitativas se traducen fácilmente en números y gráficos. Las métricas cuantitativas propuestas en esta disertación son las siguientes:

- Bytes enviados y bytes recibidos
- Tiempo de conexión
- porcentaje de error
- latencia

2.10.1 bytes enviados y bytes recibidos

La mayor parte del consumo de energía en los dispositivos se debe a la comunicación. Por lo tanto, al conocer el tamaño del paquete que es necesario para comunicarse, los dispositivos pueden administrar mejor los recursos críticos, como el nivel de la batería, y en casos avanzados, incluso pueden planificar en qué intervalos transmitir. Al analizar el tamaño del paquete, es importante recordar que las comunicaciones REST generalmente finalizan con un mensaje de respuesta (código de respuesta), por lo que el mensaje de respuesta también debe ser tenido en cuenta. Dependiendo de la situación, el tamaño del paquete puede incluso ayudar en el equilibrio de carga. Imagínese si los dispositivos usan IEEE 802.15.4, que admite velocidades de transferencia entre 20 y 250 Kbps [65] (tome en cuenta el límite superior). Si el dispositivo necesita 1000 Bytes (8000 Bits) para enviar datos, la cantidad máxima de dispositivos que se pueden conectar a una única puerta de enlace es 31. Esto supone que los dispositivos están conectados directamente a la puerta de enlace y que el número de Bytes recibidos (para confirmar que los datos se transfirieron con éxito) es menor o igual que el número de Bytes enviados.

2.10.2 Tiempo de conexión

Es importante determinar el tiempo que dura él envió de la información del sensor en bytes hacia el middleware IoT, ya que su menor valor, indicará que existe un buen canal de comunicación de extremo a extremo, y por consiguiente el middleware podrá procesar la solicitud con mayor rapidez y pueda ser mostrada a través de su dashboard (paneles) que es la interfaz del usuario final que visualiza la información en forma gráfica.

El tiempo de conexión se mide en milisegundos(ms), y valores menores a 800ms, son aceptables, para la presente investigación tenemos una distancia geográfica de más de 6,800km a los servidores de OVH, ubicados en beauharnois en Quebec, Canadá.

2.10.3 Porcentaje de error

Es un indicador que nos determinara el porcentaje de errores en el envió de información de los sensores a la plataforma middleware IoT. En la transmisión de datos cuanto mayor es la trama que se transmite, mayor es la probabilidad de que contenga algún error, en nuestro caso, las tramas son pequeñas, pero a mayor cantidad de tramas enviadas por los sensores, según el middleware, se podrían presentar errores, por lo tanto, este valor nos ayudaría a dar una calificación de cual plataforma elegir.

2.10.4 Latencia

El tiempo de respuesta se refiere a la cantidad de tiempo que un software necesita para procesar la información. El tiempo de respuesta puede ser crítico dependiendo del escenario. Por lo tanto, conocer el tiempo de respuesta del middleware es crucial, especialmente en escenarios de carga alta, donde se envía una cantidad significativa de datos a un servidor.

CAPÍTULO III

METODOLOGÍA DE LA INVESTIGACIÓN

3.1 MÉTODO DE INVESTIGACIÓN

La presente investigación es del tipo experimental tecnológico porque manipula directamente la variable independiente, las plataformas middleware, para para medir los efectos en la variable dependiente, métricas cualitativas y cuantitativas, este método se aplica con el propósito de establecer las conclusiones y generalizar los resultados de la investigación en forma cuantitativa.

3.2 DISEÑO DE LA INVESTIGACIÓN

3.2.1. Diseño

Corresponde un diseño de un solo grupo experimental (la muestra), al que se aplica una prueba antes del tratamiento experimental para observar cómo estaba al principio. Luego, se le administra el experimento y se observa el cambio producido en él mediante una prueba final (ver la siguiente figura). El lapso entre pruebas es de tres (03) horas, para prevenir cambios en el grupo:

G	O_1	X	O_2

Diagrama del diseño preprueba/posprueba

Fuente: Hernández et ál. (2014, pág. 141) [66]

Dónde:

G : Grupo de prueba

O_i : Prueba

X : Experimento

3.2.2. Tipo de investigación

La presente tesis corresponde al tipo de investigación aplicada de nivel explicativo, ésta investigación se distingue por tener propósitos prácticos inmediatos bien definidos, es decir se investiga para actuar, transformar, modificar o producir cambios en un determinado sector.

Por el propósito del estudio, el tipo de investigación es aplicado. Hernández, Fernández y Baptista [66], declaran al respecto que el objeto de esta clase de investigaciones es probar la efectividad de técnicas, instrumentos o máquinas que puedan ser utilizados inmediatamente para resolver problemas o emprender proyectos empresariales productivos. Como en este estudio, donde se pretende probar la efectividad de las plataformas middleware de IoT.

Por el criterio estratégico de investigación, el tipo al que corresponde es el experimental.

3.2.3. Nivel de investigación

Se propone una investigación a nivel exploratorio y descriptivo. Estudio exploratorio es la información general respecto a un fenómeno o problema poco conocido, incluyendo la identificación de posibles variables a estudiar en un futuro.

El estudio descriptivo es la información detallada respecto un fenómeno o problema para describir sus dimensiones (variables) con precisión. [66]

3.3. POBLACIÓN Y MUESTRA

3.3.1. Población

Está conformada por 20 middleware de código abierto (open-source).

3.3.2. Muestra

Está constituida por 2 middleware:

- Konker
- ThingsBoard

3.4. TÉCNICAS E INSTRUMENTOS

3.4.1. Técnicas

Se usa como técnica él envió de paquetes, conteniendo los parámetros y números de usuarios, simultáneamente que acceden a la plataforma.

3.4.2. Instrumentos:

Corresponde aplicar una prueba de performance de la plataforma middleware, ya que permite conocer y mitigar el desempeño del mismo.

En los valores por defecto para petición HTTP debemos parametrizar en el Apache JMeter:

- Nombre del servidor o IP: ${__P(nServidor)}
- Puerto: ${__P(nPuerto)}
- Protocolo: ${__P(protocolo)}

CAPÍTULO IV

RESULTADOS Y PRUEBAS

4.1. HERRAMIENTAS PARA PRUEBAS DE CARGA

Tomamos como referencia el cuadro elaborado por Araujo,Victor [3] y realizamos una actualización de los plugin, en el caso de la herramienta Apache JMeter y Gatling, esta última es usada por el Middleware ThingsBoard.io para pruebas del protocolo MQTT, así también actualizamos la valoración de la extensibilidad que viene hacer la capacidad que tiene la herramienta para poder adicionar plugin incrementamos su valoración a la Herramienta Gatling.

Tabla 5 Herramientas populares de prueba de carga de código abierto

Herramienta	Lenguaje	HTTP	MQTT	CoAP	Extensibilidad
Tsung	Erlang	Nativo	Nativo	No	+
Apache JMeter	Java	Nativo	Plugin [67]	No	+++
Gatling	Scala	Nativo	Plugin [68]	No	+++
Locust	Python	Nativo	No	No	++
httperf	C	Nativo	No	No	++

Fuente: [3]

4.2. APACHE JMETER

La aplicación Apache JMeter™ es un software de código abierto, una aplicación Java 100% pura diseñada para cargar el comportamiento funcional de la prueba y medir el rendimiento. Originalmente fue diseñado para probar aplicaciones web, pero desde entonces se ha expandido a otras funciones de prueba [69].

Figura 33 logo Apache JMeter

Fuente: [69].

El motor JMeter puede generar carga utilizando diversos protocolos y lógica compleja que el usuario puede personalizar a través de la definición de un plan de prueba. Un plan de prueba describe la serie de pasos que JMeter ejecutará durante la prueba. JMeter proporciona un conjunto de abstracciones llamadas elementos de prueba para describir estos pasos. Los elementos de prueba describen los protocolos, los tipos de solicitudes a usar, su configuración y la lógica que guía la ejecución de la prueba.

Uno de los elementos de prueba de más alto nivel de JMeter es el grupo de hilos. La arquitectura de JMeter tiene múltiples subprocesos, y en un escenario típico, un subproceso se usará para modelar a un usuario mediante la realización de las acciones definidas en el plan de prueba independientemente de otros subprocesos. Los controladores lógicos y los muestreadores, los elementos que envían solicitudes, se agregan a los grupos de subprocesos. Los grupos de subprocesos se pueden configurar con el número de subprocesos totales que se generan, la cantidad de tiempo durante el cual se engendrarán estos subprocesos y el número

de veces que se repiten las acciones definidas en el grupo de subprocesos. Cuando se realizan pruebas funcionales y no funcionales, el número de subprocesos y el tiempo necesario para generarlos debe regularse para que coincida con la carga deseada.

Los controladores lógicos se pueden agregar dentro de un grupo de hilos para personalizar las condiciones que utiliza JMeter para decidir cuándo ejecutar un muestreador u otro componente. Los controladores lógicos incluyen reglas que pueden ejecutar otros componentes de una sola vez, al azar, una cantidad fija de veces o cuando sea necesario para lograr un rendimiento particular. Esto le permite al usuario de JMeter definir escenarios complejos.

Los muestreadores son los elementos de prueba que envían solicitudes al sistema que se está probando. JMeter incluye muestreadores para varios protocolos, incluidos FTP, HTTP, JDBC, TCP, LDAP, JMS y otros. Es posible extender JMeter y definir muestreos para otros protocolos que no son parte de la distribución en sentido ascendente. Los muestreadores son configurables de acuerdo con el protocolo y su comportamiento puede ser ajustado por otros componentes.

Los temporizadores se pueden agregar como elementos secundarios de muestreadores y controladores para pausar, sincronizar o ajustar su comportamiento. Por lo tanto, los temporizadores se utilizan para dar forma a la distribución de las solicitudes y su tasa de generación. Los temporizadores varían desde simples pausas constantes de tiempo hasta comportamientos que siguen una distribución estadística particular, incluidas distribuciones gaussianas, de Poisson y uniformes. Otro uso importante de los temporizadores es sincronizar los subprocesos para ejecutar un muestreador o un controlador al mismo tiempo, creando efectivamente solicitudes concurrentes al sistema bajo prueba.

Para parametrizar los scripts de prueba, JMeter proporciona elementos de configuración y variables. Los elementos de configuración definen un conjunto de valores que se aplican a elementos de prueba específicos en una parte del plan de prueba. Por ejemplo, se puede agregar un elemento de configuración que define el host al destino para todos los muestreadores HTTP. Las variables también pueden

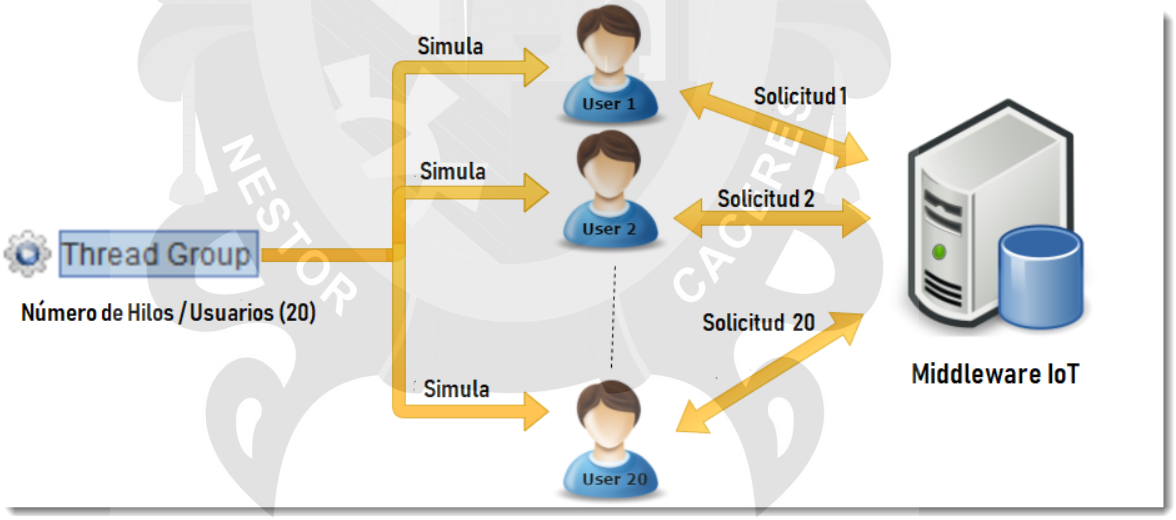
ser definidas por el usuario y se puede acceder a ellas en cada hilo. JMeter expone un conjunto de variables predefinidas a las que se puede acceder durante la prueba para configurar parámetros y proporcionar una introspección sobre el estado actual de la prueba.

4.2.1 Thread Group

Los grupos de hilos son los pasos iniciales del plan de pruebas de JMeter. Se pueden definir como el número de hilos(usuarios) en un Thread Group. Cada hilo simula un usuario real que solicita al servidor bajo una prueba.

Si se establece el número de hilos en 20; JMeter creará y simulará 20 usuarios virtuales durante la prueba de carga. En el siguiente diagrama puede ayudarnos a entender mejor.

Figura 34 Ejemplo de creación de 20 usuarios en un Thread Group



Fuente: Elaboración propia

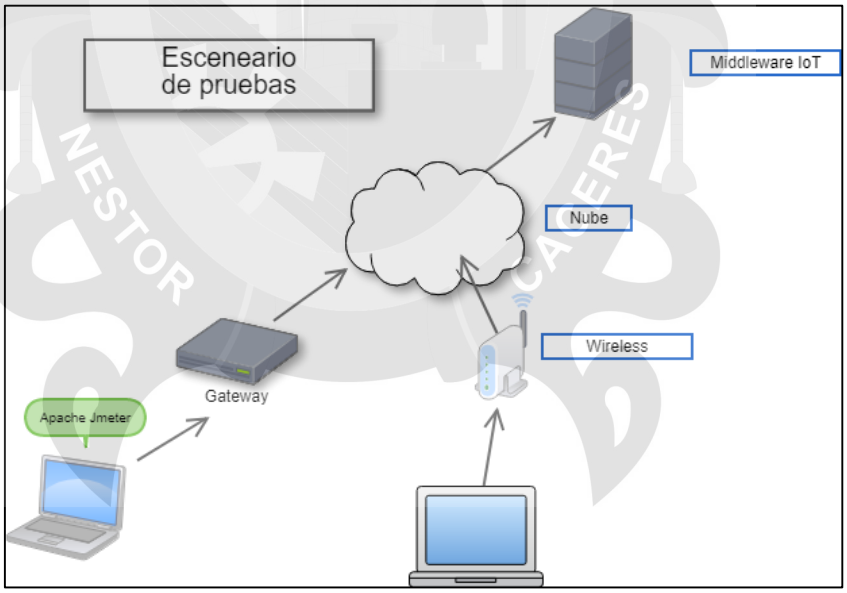
4.3 ESCENARIO DE PRUEBAS

El escenario estará constituido por el JMeter que reemplazara a los sensores, donde los dispositivos intercambian información y envían datos a un middleware.

Posteriormente, los usuarios externos o las aplicaciones pueden acceder a los datos (desde cualquier lugar del mundo). Por esta razón, es esencial que el servidor almacene continuamente datos que se pueden recuperar en cualquier momento y en cualquier lugar con las credenciales adecuadas. Una ventaja de las soluciones PaaS proviene del hecho de que están ubicadas en la nube y los usuarios autenticados pueden acceder a los datos ubicados en el servidor desde cualquier lugar de Internet, sin tener que preocuparse por implementar o administrar la infraestructura [70].

Las plataformas de middleware que se evalúan en este capítulo son de código abierto. El primer paso del estudio de evaluación de desempeño considerará una comparación cualitativa entre las soluciones disponibles para filtrar qué plataformas de middleware son útiles para el escenario elegido. La evaluación cualitativa será seguida por un estudio de comparación cuantitativa para evaluar objetivamente las soluciones.

Figura 35 Escenario de pruebas



Fuente: Elaboración propia

4.4 EVALUACIÓN DE DESEMPEÑO CONSIDERANDO MÉTRICAS CUALITATIVAS

Los aspectos más importantes de la comparación cualitativa que se muestran en la Tabla 5 Clasificación de las Métricas Cuantitativas y Cualitativas según SMI son las siguientes: usabilidad, aprendizaje, seguridad. Con respecto a las características de seguridad descritas en las métricas cualitativas, solo se implementa la autenticación por dispositivo (y solo en algunos middlewares). Según el mejor conocimiento de los autores, incluso las soluciones que están disponibles en la nube en forma de PaaS no implementan las otras características de seguridad sugeridas.

4.5 EVALUACIÓN DEL RENDIMIENTO UTILIZANDO MÉTRICAS CUANTITATIVAS

Los experimentos presentes en este capítulo se realizaron en un escenario de red real. Los paquetes se generaron a través de Apache Jmeter [69] en una computadora con acceso vía wifi como sería el caso de un sensor y luego esta conexión se conecta a una red cableada que en ruta nuestros paquetes a internet.
Una LAN Wi-Fi garantiza que los paquetes puedan encontrar dificultades como son la calidad de la señal, distancia al punto de acceso inalámbrico y el retraso o la pérdida de paquetes.
Para la evaluación cuantitativa, se ha considerado las plataformas Konker de origen brasileño y ThingsBoard Inc. de origen estadounidense. La elección de la plataforma se ha debido a su facilidad mediana de instalación en un servidor, hemos encontrado dificultades en instalar otras plataformas como kaa, Sitewhere, orion, algunas de ellas con poca documentación para su instalación y otras solo contenían una imagen virtualizada para instalarlo en un ambiente local, cosa que no cumpliría nuestros objetivos de realizarlo en un escenario real.

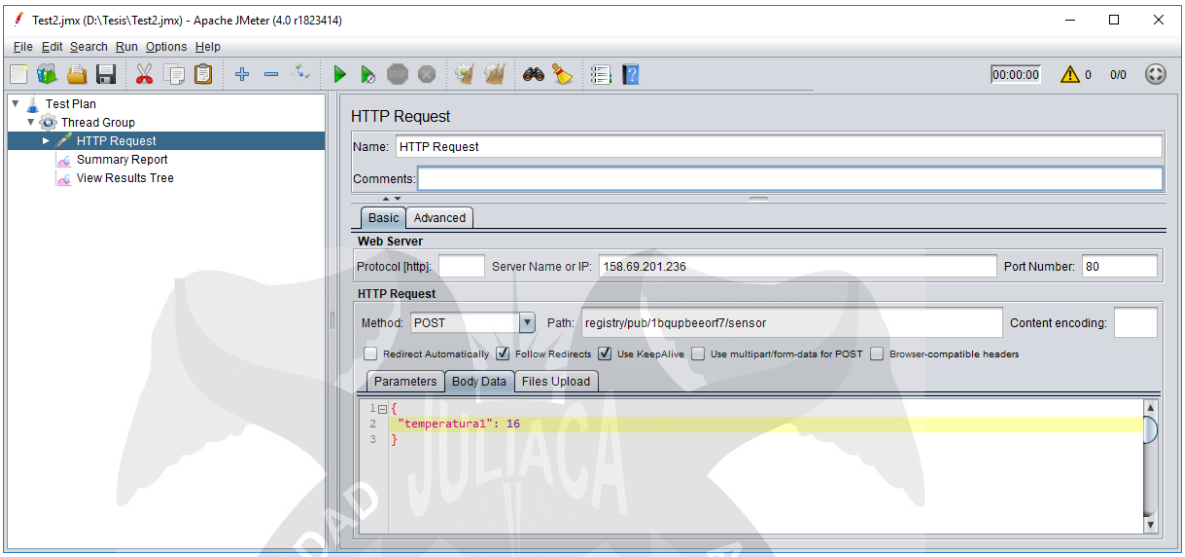


Figura 36 Interfaz de Apache Jmeter con los parámetros de prueba.

Los sensores envían varios resultados de variables al middleware, lo que significa que un objeto puede enviar la temperatura externa, humedad, su ubicación actual, índice de radiación UV, el nivel de batería y mucho más. Estas variables también se conocen como parámetros.

Para las pruebas usamos el servicio de OVH, que es una compañía que cuenta con 300 mil servidores, 27 centros de datos, ubicados en 19 países, tiene más de 18 millones de aplicaciones web alojadas, usamos el centro de datos de Beauharnois en Canadá. [71]

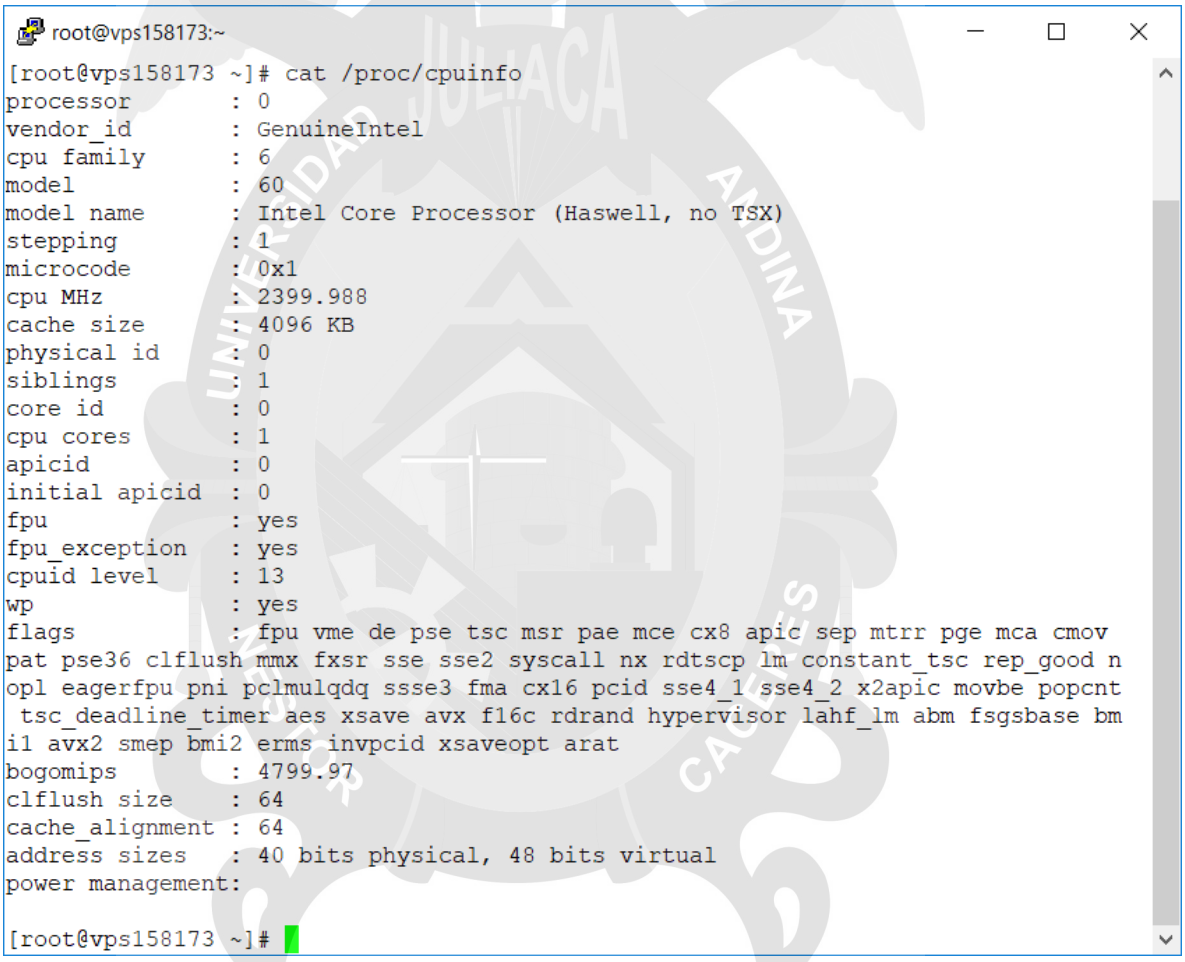
Desplegamos un servidor virtual privado (VPS, por sus siglas en inglés) es un servidor dedicado virtualizado. A diferencia de un alojamiento web (compartido), cuya gestión técnica recae en el proveedor, nosotros somos responsable de la administración del VPS.

Los detalles sobre las especificaciones de hardware de la máquina host, se pueden encontrar en la Tabla 6. La información sobre el sistema operativo que ejecutó cada middleware se puede encontrar en la Tabla 7.

Tabla 6 Especificaciones de hardware del Servidor

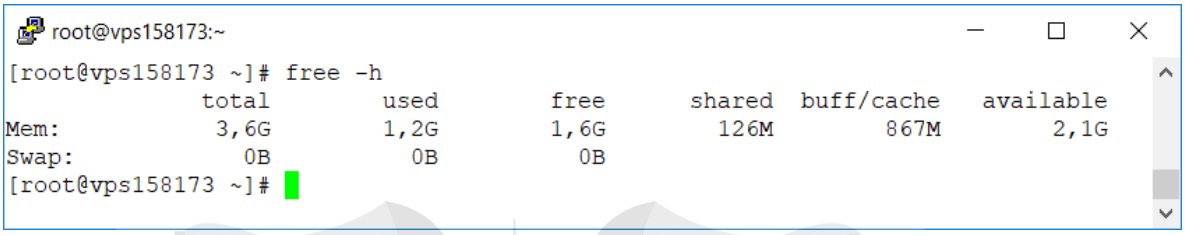
Item	Descripción
Procesador	Intel Core
Frecuencia	2.4 GHz
Ram	4GB
Disco Duro	40 GB SSD

Figura 37 Información del CPU del VPS



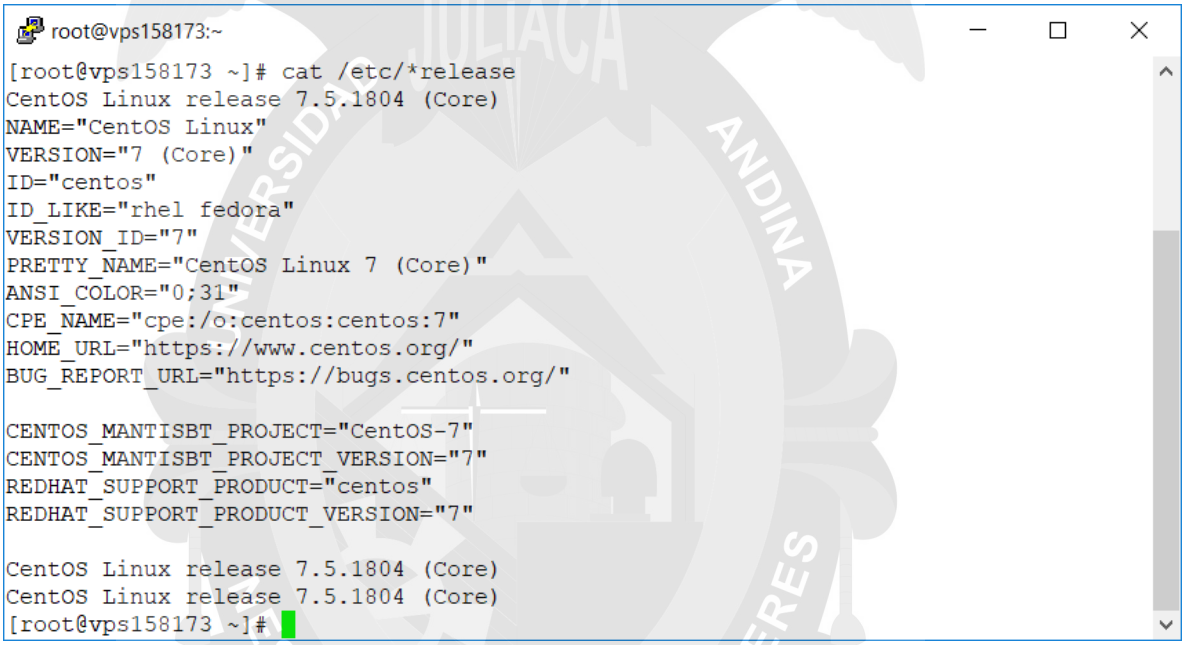
Fuente: Elaboración propia

Figura 38 Información de la Memoria del VPS



Fuente: Elaboración propia

Figura 39 Información del sistema operativo del middleware Konker en el VPS



Fuente: Elaboración propia

Figura 40 Información del Sistema Operativo del middleware ThingsBoard en el VPS

Fuente: Elaboración propia

Tabla 7 Sistema operativo que está instalado en cada Middleware

Sistema Operativo	Middleware
Centos 7.5.18.04	Konker
Ubuntu 16.04.5 LTS	ThingsBoard

Fuente: Elaboración propia

4.6 MÉTRICA: BYTES ENVIADOS Y BYTES RECIBIDOS

Los bytes enviados y recibidos se analizan porque los métodos RESTFUL generalmente finalizan con un código de respuesta. Este experimento es crítico, porque ayuda a equilibrar la cantidad de dispositivos por puerta de enlace. El middleware que se desempeña mejor estará determinado por una suma de Bytes enviados y recibidos.

La prueba tendrá el siguiente detalle para lo cual se deberán crear archivos independientes del apache JMeter, estos llevan la extensión .jmx y lo detallamos en el siguiente cuadro.

Tabla 8 Nombres de Archivos de JMeter indicando el middleware y los parámetros

Middleware	Parámetros	Nombre de archivo
Konker	1	Test-Konker-u1-p1.jmx
	5	Test-Konker-u1-p5.jmx
	10	Test-Konker-u1-p10.jmx
ThingsBoard	1	Test-Thingsboard-u1-p1.jmx
	5	Test-Thingsboard-u1-p5.jmx
	10	Test-Thingsboard-u1-p10.jmx

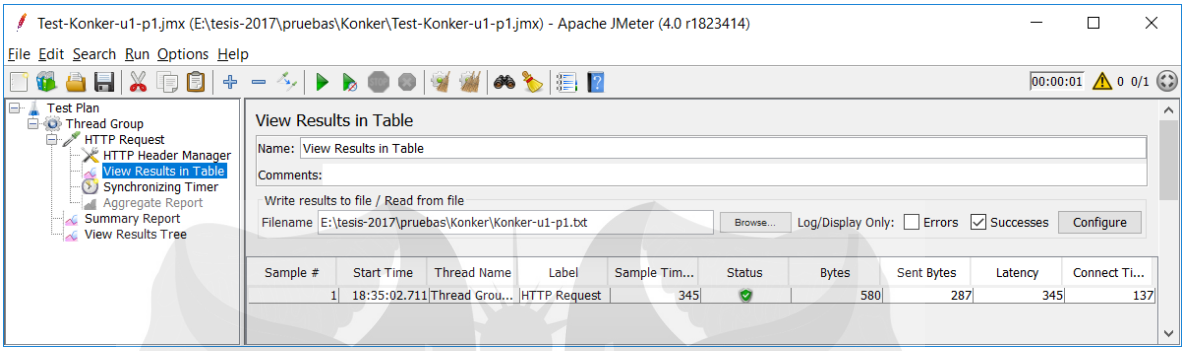
Fuente: Elaboración propia

Para realizar esta prueba, debemos configurar el Apache JMeter con la dirección IP del Middleware IoT a probar, el número de usuarios, el número de parámetros, que será enviado al servidor. La prueba es real y no se utiliza un ambiente virtualizado o de laboratorio. A continuación mostramos los resultados de cada middleware considerando el número de parámetros 1, 5 y 10, se ha escogido este universo de parámetros, justificando que por cada estación compuesta por sensores, estos pueden enviar hasta un máximo de 10 mediciones. Como es el caso de aplicaciones de agricultura de precisión, los valores podrían ser: ubicación, temperatura, humedad, velocidad del viento, presión barométrica, índice de radiación, humedad del suelo y lluvia.

4.6.1 Plataforma konker con 1 parámetro

El resultado que nos envía el JMeter es de 287 bytes enviados y de 580 bytes recibidos, como se aprecia en la Figura.

Figura 41 Resultados de bytes enviados y recibidos en Konker con 1 parámetro

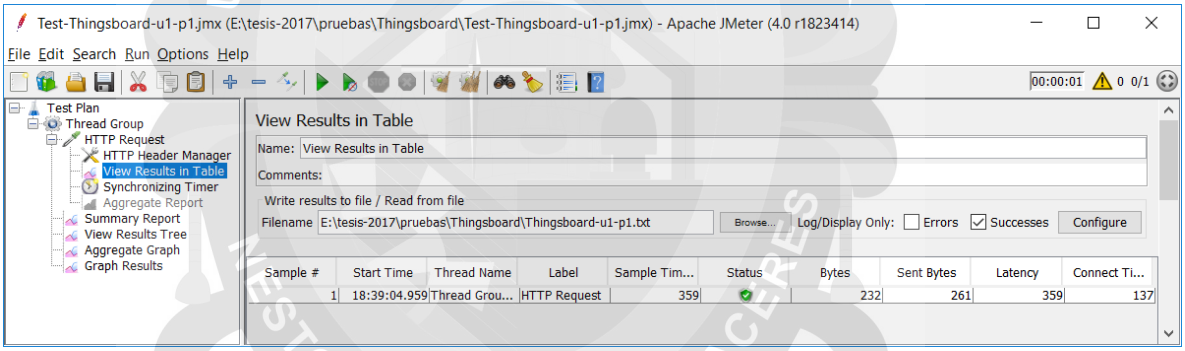


Fuente: Elaboración propia

4.6.2 Plataforma thingsboard con 1 parámetro

Al realizar la prueba con un parámetro obtenemos como resultado la cantidad de 260 bytes enviados y de 232 bytes recibidos, tal como mostramos en la Figura.

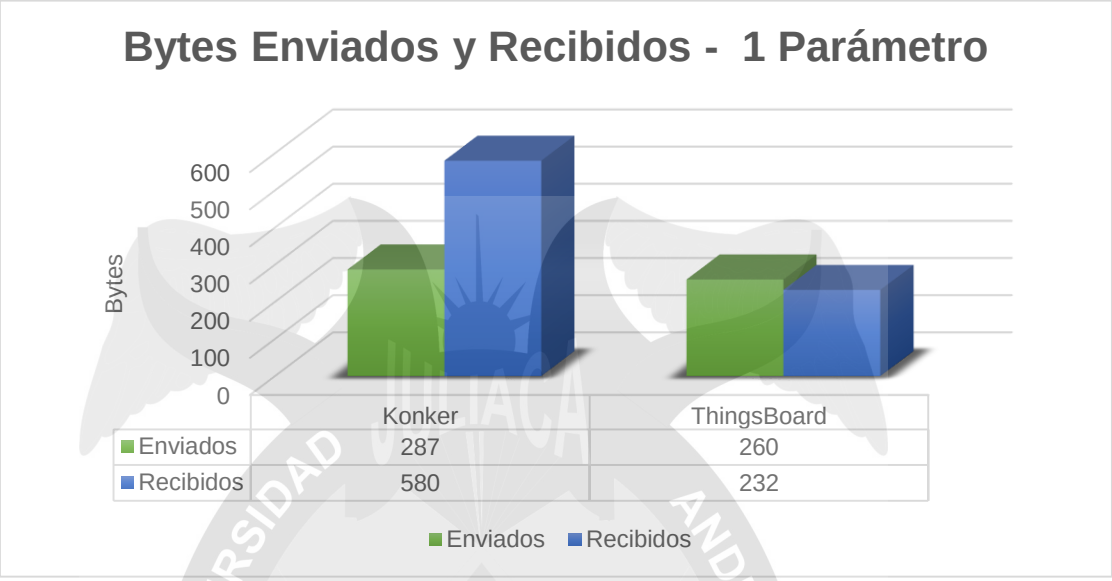
Figura 42 Resultados de bytes enviados y recibidos en ThingsBoard con 1 parámetro



Fuente: Elaboración propia

La Figura 43 se presenta el tamaño del paquete de una única solicitud REST para publicar datos con 1 parámetros, del cual concluimos que el middleware ThingsBoard, es el que tiene mejor desempeño, al tener la menor cantidad de bytes que se envían como se reciben con la respuesta, la diferencia entre bytes enviados del middleware Konker y ThingsBoard es de apenas de 27 bytes, en cambio de los bytes recibidos la diferencia es de 348 bytes, es cantidad representaría más del doble de bytes recibidos del middleware ThingsBoard.

Figura 43 Bytes enviados y recibidos con un solo parámetro

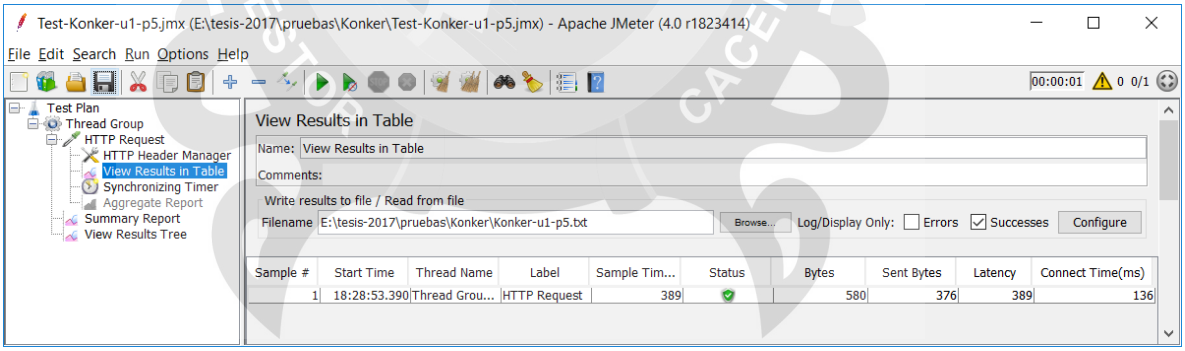


Fuente: Elaboración propia

4.6.3 Middleware konker con 5 parámetros

En la siguiente prueba, debemos configurar el apache JMeter con 5 parámetros, al ejecutar la prueba, el resultado que nos envía es de 376 bytes enviados y de 580 bytes recibidos.

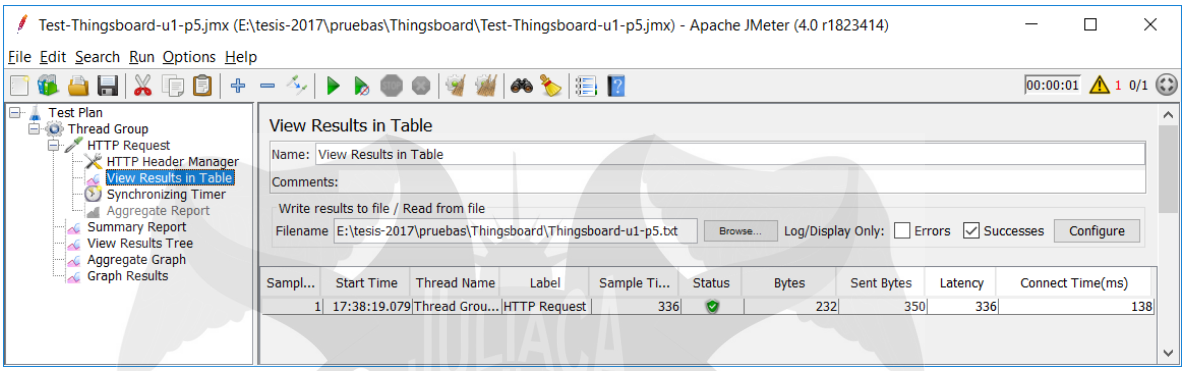
Figura 44 Resultados de bytes enviados y recibidos en Konker con 5 parámetros



Fuente: Elaboración propia

4.6.4 Middleware thingsboard con 5 parámetros

Figura 45 Resultados de bytes enviados y recibidos en ThingsBoard con 5 parámetros

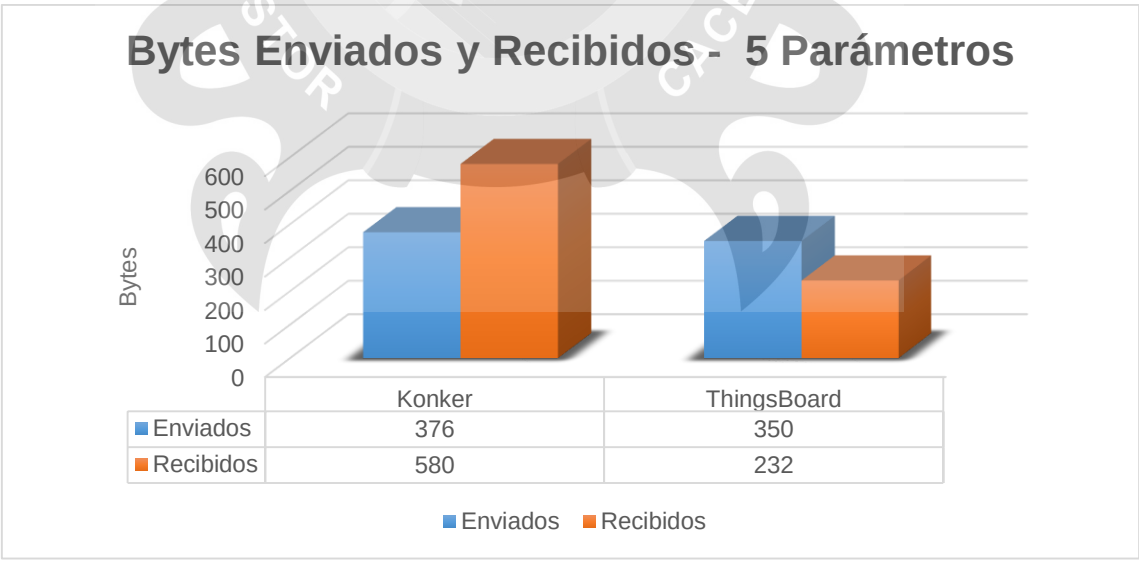


Fuente: Elaboración propia

4.6.5 Cuadro comparativo con 5 parámetros

La Figura 46. concluimos que el middleware ThingsBoard, es el que tiene mejor desempeño, al tener la menor cantidad de bytes que se envían como se reciben, la diferencia entre bytes enviados del middleware Konker y ThingsBoard es de apenas de 26 bytes, en cambio de los bytes recibidos la diferencia es de 348 bytes, es cantidad representaría más del doble de bytes recibidos del middleware ThingsBoard.

Figura 46 Bytes enviados y recibidos con cinco parámetros

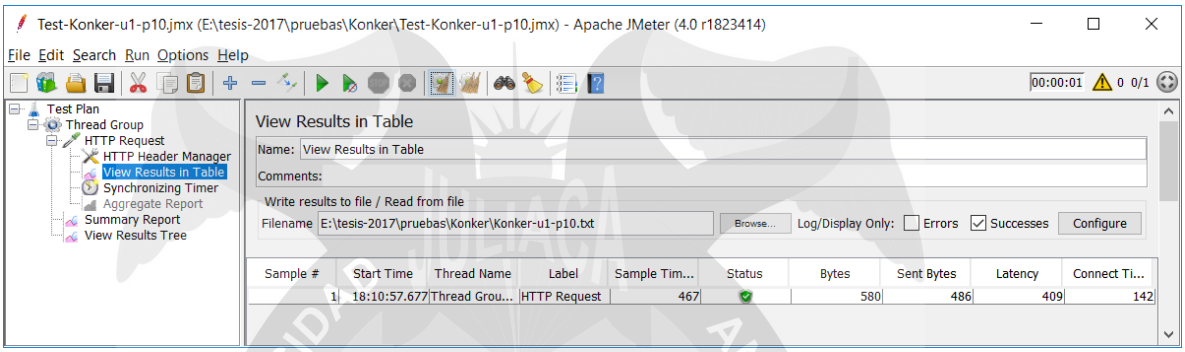


Fuente: Elaboración propia

4.6.6 Middleware konker con 10 parámetros

La cantidad de bytes enviados es de 486 y la de recibidos es de 580, como mostramos en figura.

Figura 47 Resultados de bytes enviados y recibidos en Konker con 10 parámetros

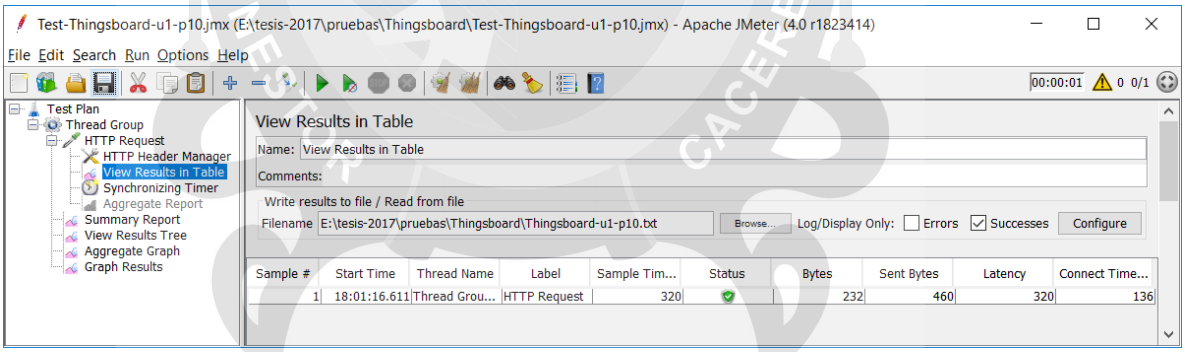


Fuente: Elaboración propia

4.6.7 Middleware thingsboard con 10 parámetros

La cantidad de bytes enviados es de 460 y la de recibidos es de 232, como mostramos en figura.

Figura 48 Resultados de bytes enviados y recibidos en ThingsBoard con 5 parámetros



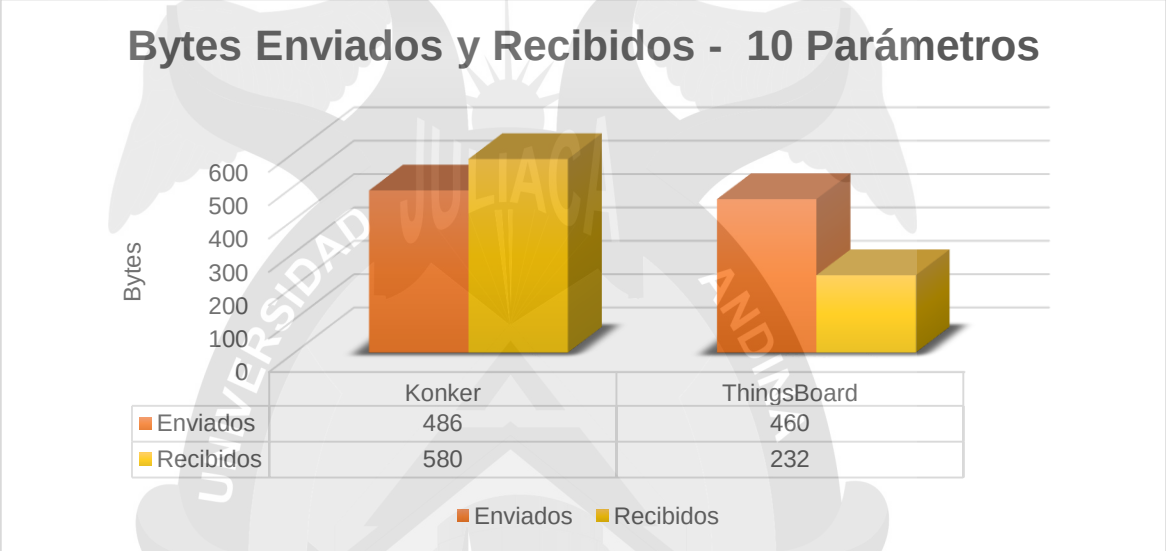
Fuente: Elaboración propia

4.6.8 Cuadro comparativo con 10 parámetros

En la Figura concluimos que el middleware ThingsBoard, es el que tiene mejor desempeño, al tener la menor cantidad de bytes que se envían como se reciben, la

diferencia entre bytes enviados del middleware Konker y ThingsBoard es de apenas de 26 bytes, en cambio de los bytes recibidos la diferencia es de 348 bytes, es cantidad representaría más del doble de bytes recibidos del middleware ThingsBoard.

Figura 49 Bytes enviados y recibidos con diez parámetros



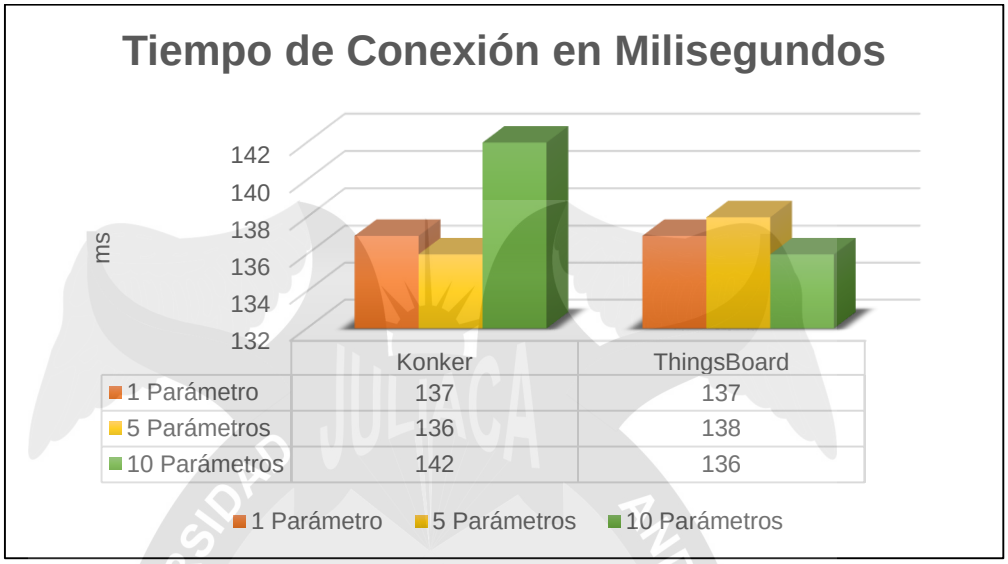
Fuente: Elaboración propia

4.7 MÉTRICA: TIEMPO DE CONEXIÓN (MS)

JMeter mide el tiempo que tomó establecer la conexión, incluido el protocolo de enlace SSL. Tenga en cuenta que el tiempo de conexión no se resta automáticamente de la latencia. En caso de error de conexión, la métrica será igual al tiempo que tomó enfrentar el error, por ejemplo, en el caso de un tiempo de espera, debería ser igual al tiempo de espera de la conexión.

Es la medición que obtenemos del JMeter, para cada uno de los Middleware IoT, Konker y ThingsBoard, se ha tomado este valor según la cantidad de parametros.

Figura 50 Tiempo de conexión en Milisegundos todos los parametros



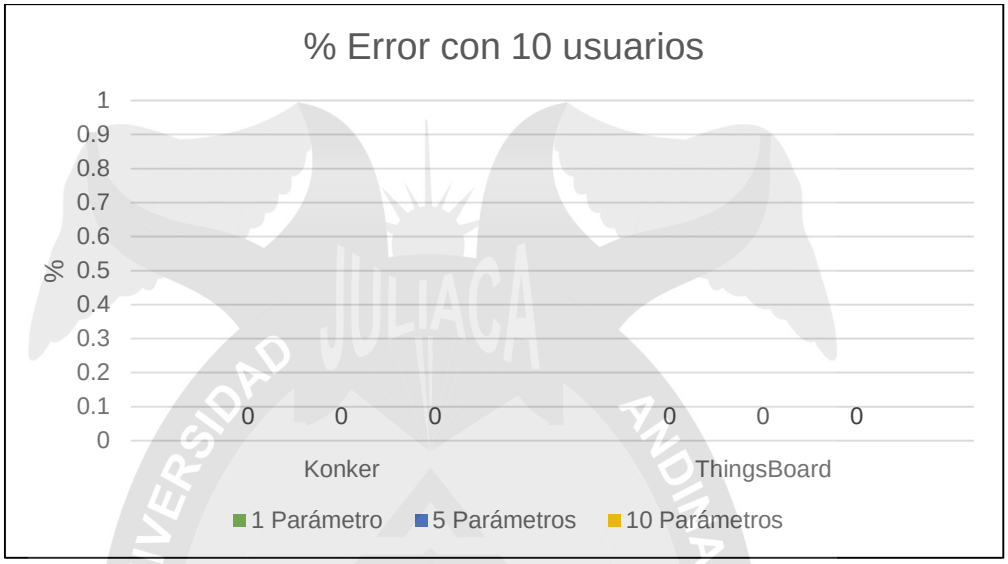
Fuente: Elaboración propia

4.8 MÉTRICA: PORCENTAJE DE ERROR

El objetivo de este experimento es determinar si las plataformas de middleware pueden manejar la carga entrante sin errores. Para lograr este objetivo, los datos se enviaron cuando estaban presentes 10, 500 y 1000 usuarios simultáneos. En cada experimento, los usuarios enviaban 1, 5 y 10 parámetros.

4.8.1 Porcentaje de error con 10 usuarios

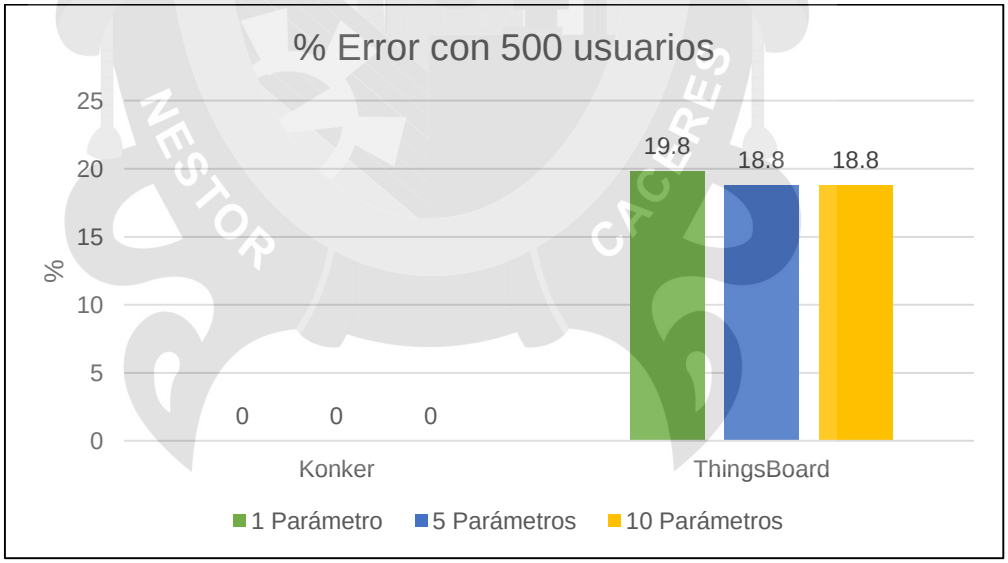
Figura 51 Porcentaje de error con 10 usuarios



Fuente: Elaboración propia

4.8.2 Porcentaje de error con 500 usuarios

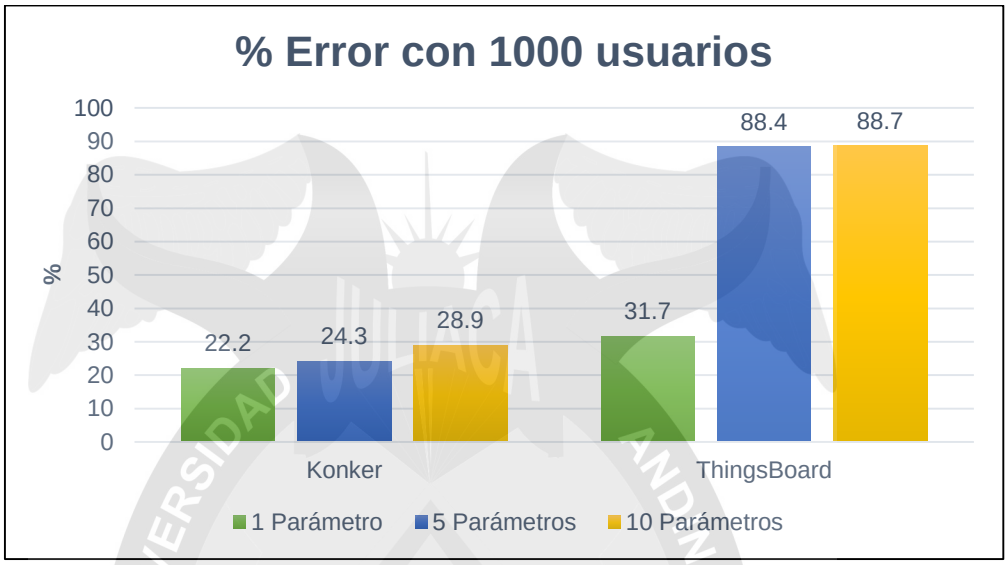
Figura 52 Porcentaje de error con 500 usuarios



Fuente: Elaboración propia

4.8.3 Porcentaje de error con 1000 usuarios

Figura 53 Porcentaje de error con 1000 usuarios



Fuente: Elaboración propia

4.9 MÉTRICA: LATENCIA

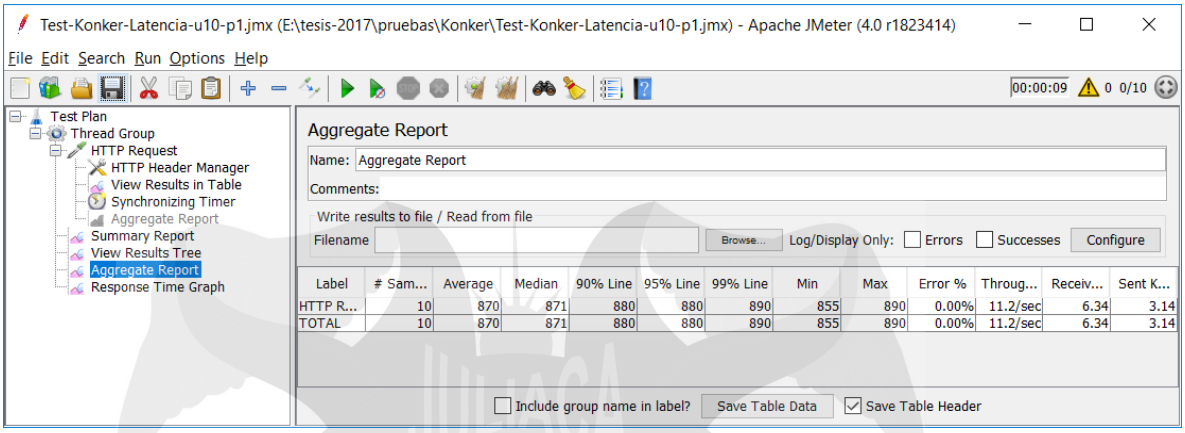
El objetivo de este experimento es determinar la cantidad de tiempo de retardo de cada plataforma de middleware. Para lograr este objetivo, los datos se enviaron con 1, 5 y 10 parámetros, cuando hubo 10, 500 y 1000 usuarios concurrentes. Para esta prueba, las medidas estadísticas más importantes son el promedio y la mediana. La mediana se ve favorecida con respecto a la desviación estándar porque, dado que fue un experimento de la vida real, y no una simulación, es normal que el servidor atienda algunas de las solicitudes en un tiempo extremadamente alto.

4.9.1 Latencia con 10 usuarios

4.9.2 Middleware konker con 1 parámetro y 10 Usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 871ms.

Figura 54 Resultados de la latencia del Middleware konker con 1 parámetro y 10 usuarios

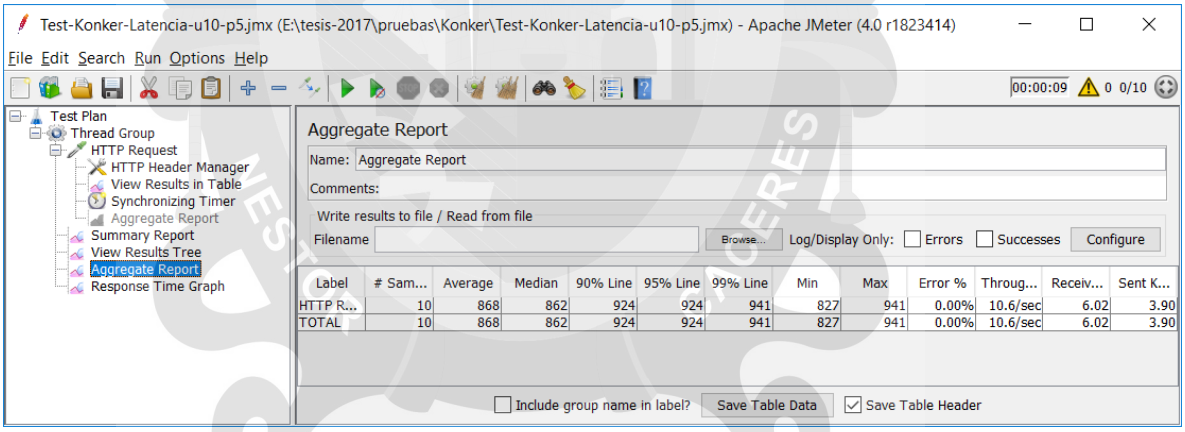


Fuente: Elaboración propia

4.9.3 Middleware konker con 5 parámetros y 10 Usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 862ms.

Figura 55 Resultados de la latencia del Middleware konker con 5 parámetros y 10 usuarios

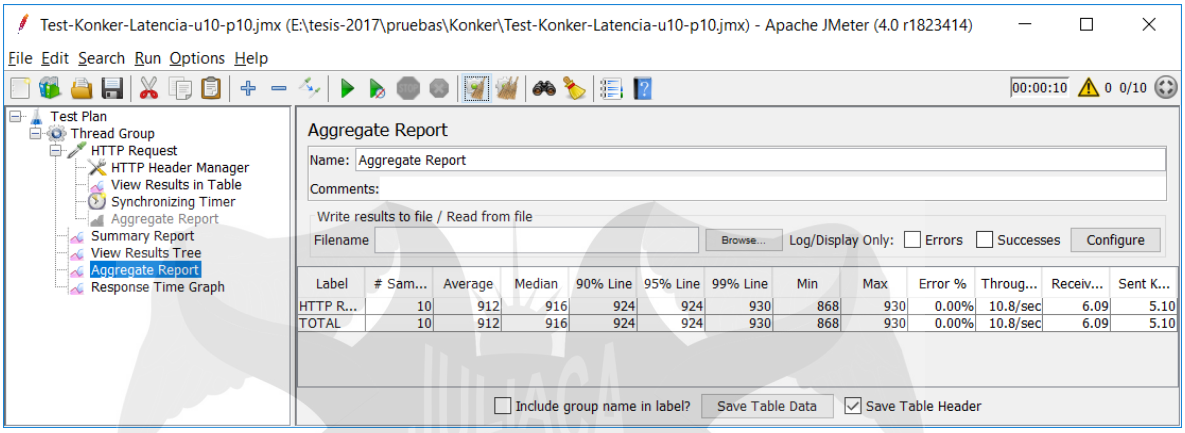


Fuente: Elaboración propia

4.9.4 Middleware konker con 10 parámetros y 10 Usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 916ms.

Figura 56 Resultados de la latencia del Middleware konker con 10 parámetros y 10 usuarios

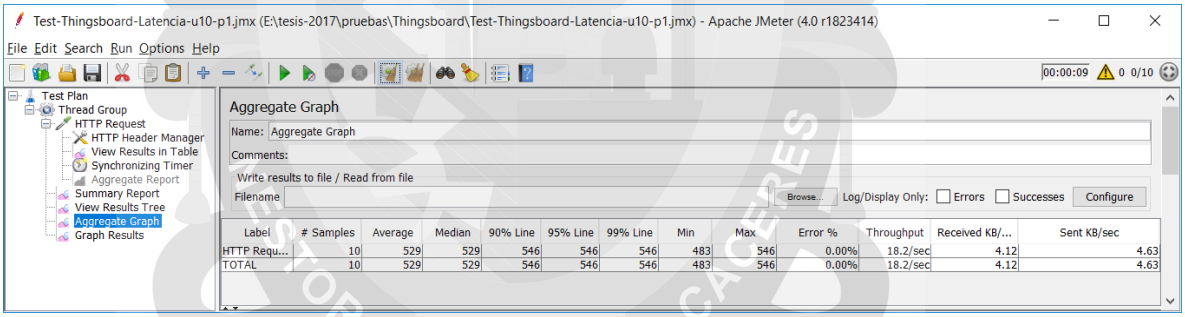


Fuente: Elaboración propia

4.9.5 Middleware thingsboard con 1 parámetro y 10 usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 529ms.

Figura 57 Resultados de la latencia del Middleware ThingsBoard con 1 parámetro y 10 usuarios

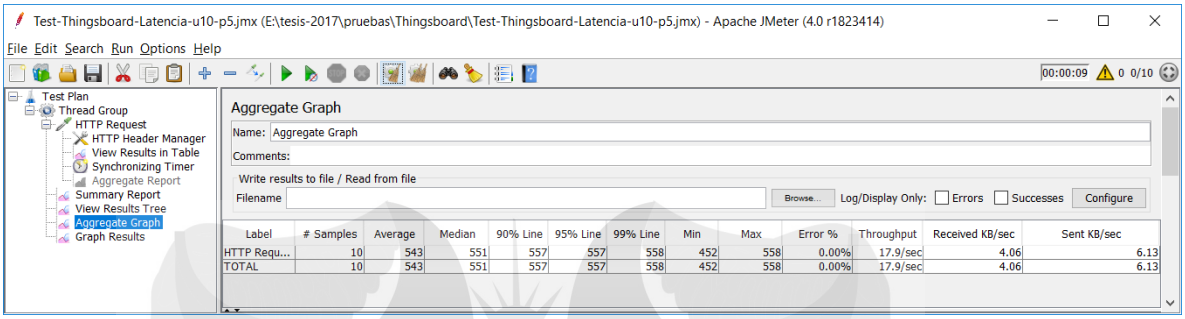


Fuente: Elaboración propia

4.9.6 Middleware thingsboard con 5 parámetros y 10 usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 551ms.

Figura 58 Resultados de la latencia del Middleware ThingsBoard con 5 parámetros y 10 usuarios



Fuente: Elaboración propia

4.9.7 Middleware thingsboard con 10 parámetros y 10 usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 627ms.

Figura 59 Resultados de la latencia del Middleware ThingsBoard con 10 parámetros y 10 usuarios

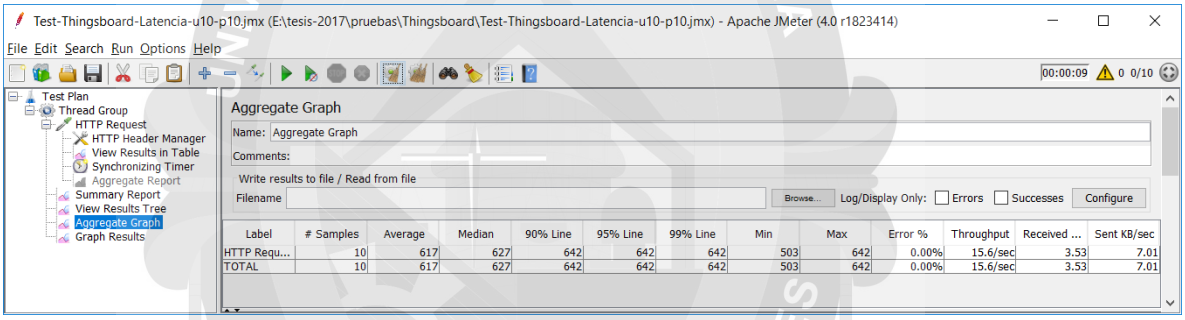
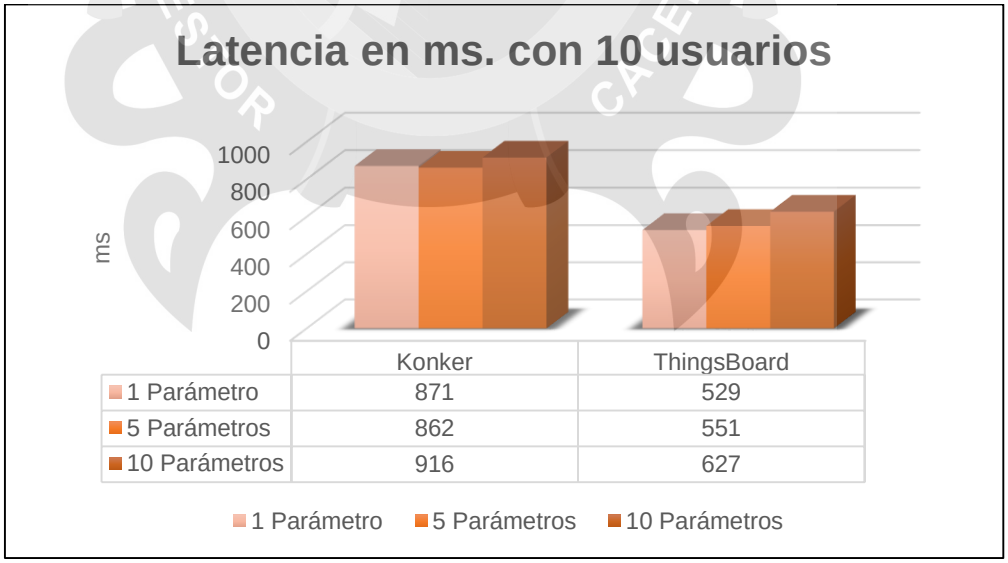


Figura 60 Gráfico de la Latencia con 10 Usuarios



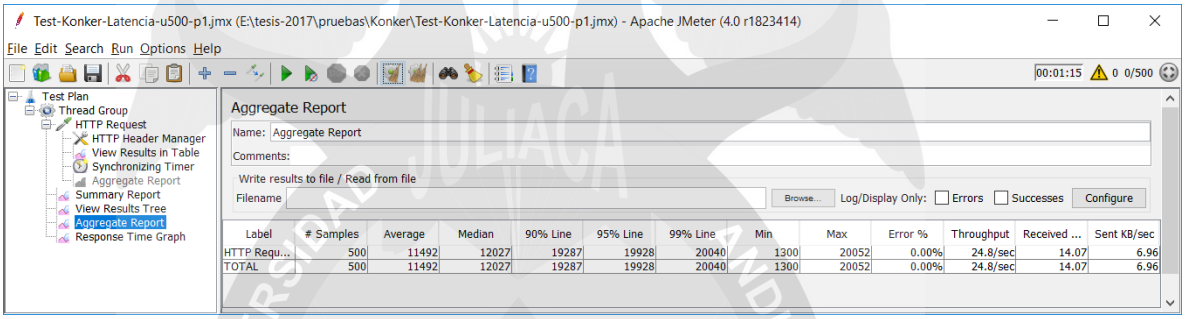
Fuente: Elaboración propia

4.9.8 Latencia con 500 usuarios

4.9.9 Middleware konker con 1 parámetro y 500 Usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 12027 ms.

Figura 61 Resultados de la latencia del Middleware konker con 1 parámetro y 500 usuarios

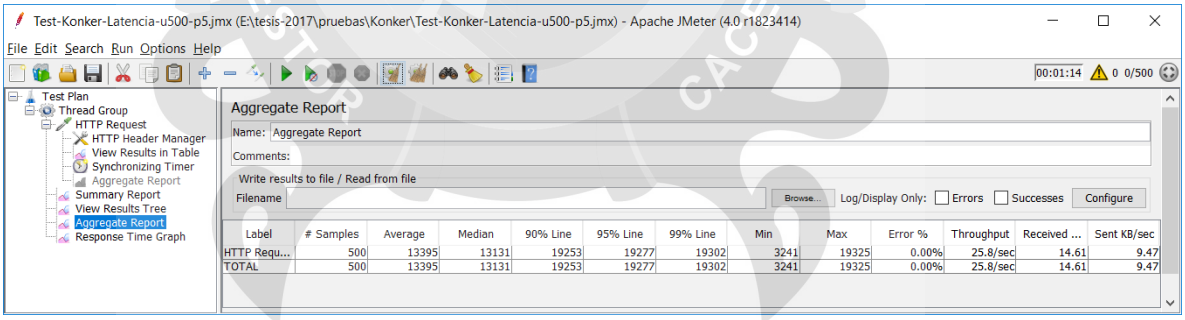


Fuente: Elaboración propia

4.9.10 Middleware konker con 5 parámetros y 500 Usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 13131 ms.

Figura 62 Resultados de la latencia del Middleware konker con 5 parámetros y 500 usuarios

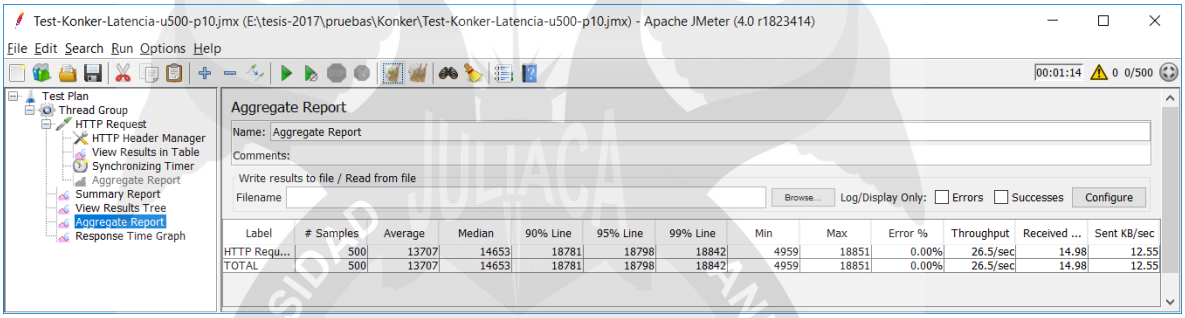


Fuente: Elaboración propia

4.9.11 Middleware konker con 10 parámetros y 500 Usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 14653 ms.

Figura 63 Resultados de la latencia del Middleware konker con 10 parámetros y 500 usuarios

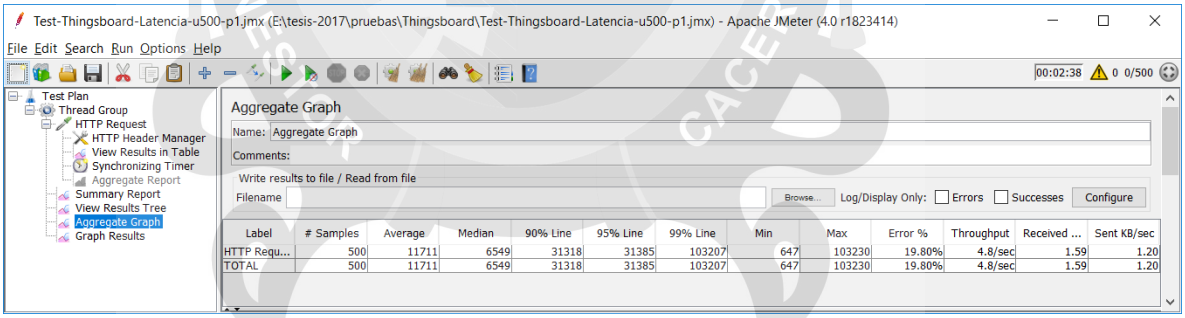


Fuente: Elaboración propia

4.9.12 Middleware thingsBoard con 1 parámetro y 500 usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 6549 ms.

Figura 64 Resultados de la latencia del Middleware ThingsBoard con 1 parámetro y 500 usuarios

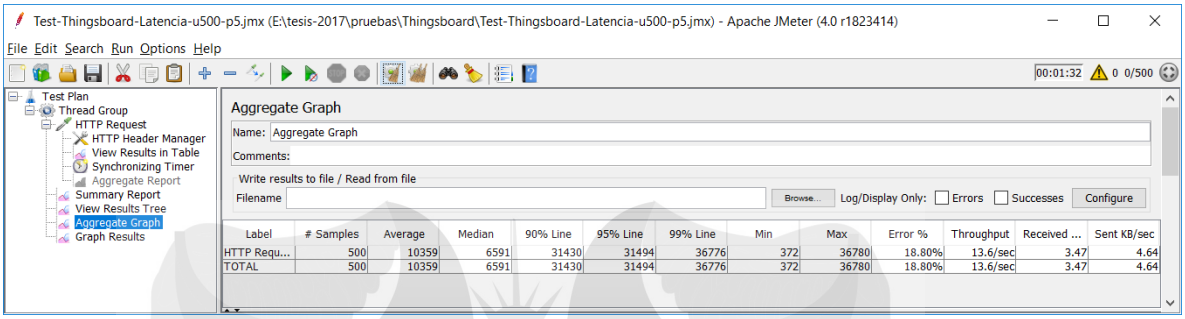


Fuente: Elaboración propia

4.9.13 Middleware thingsboard con 5 parámetros y 500 usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 6591 ms.

Figura 65 Resultados de la latencia del Middleware ThingsBoard con 5 parámetros y 500 usuarios

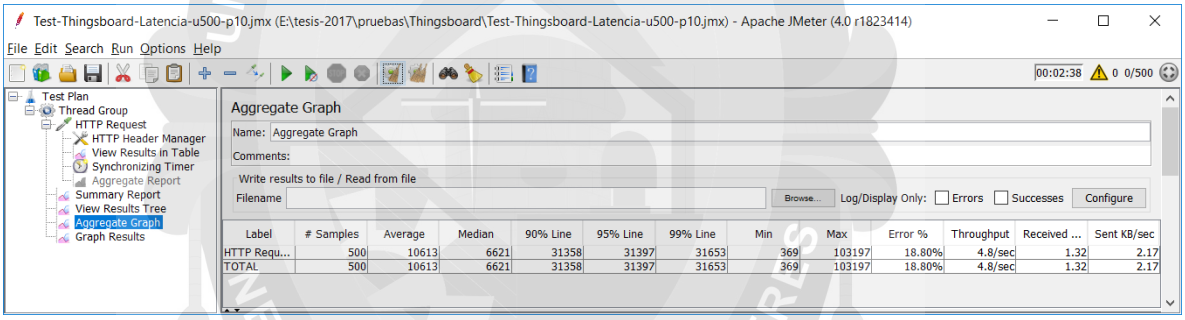


Fuente: Elaboración propia

4.9.14 Middleware thingsboard con 10 parámetros y 500 usuarios

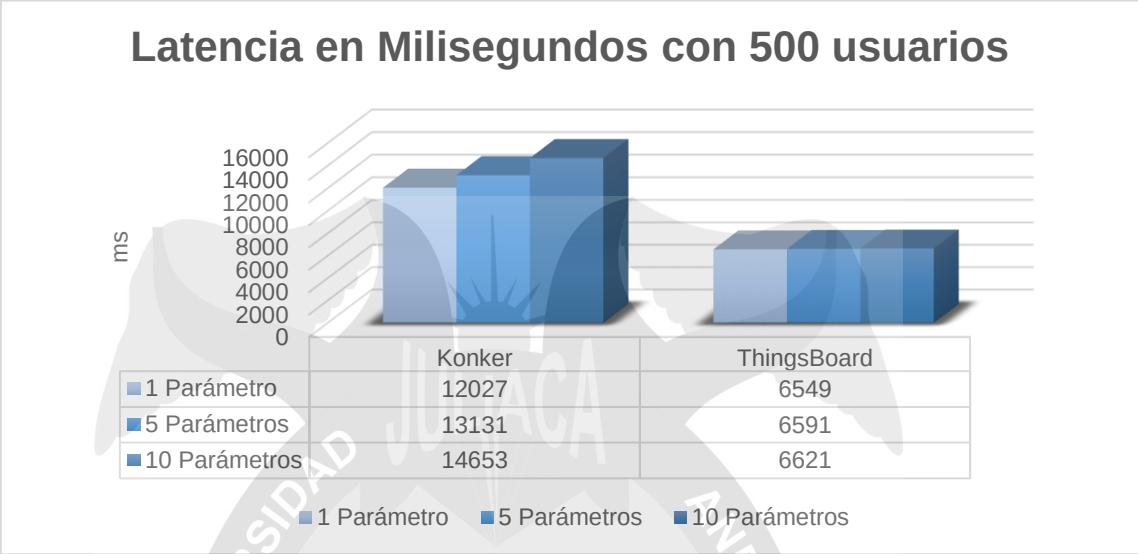
De la figura, se toma la Columna de la Mediana de la Latencia, que es de 6621 ms.

Figura 66 Resultados de la latencia del Middleware ThingsBoard con 10 parámetros y 500 usuarios



Fuente: Elaboración propia

Figura 67 Gráfico de la Latencia con 500 Usuarios



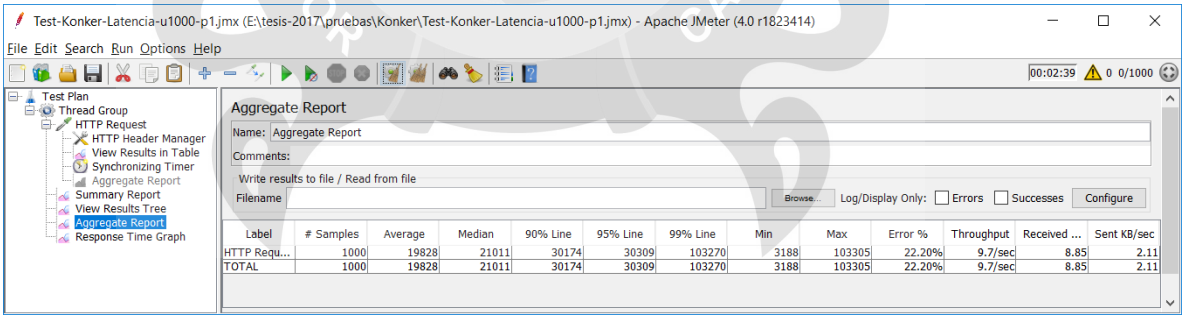
Fuente: Elaboración propia

4.9.15 Latencia con 1000 Usuarios

4.9.16 Middleware konker con 1 parámetro y 1000 usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 21011 ms.

Figura 68 Resultados de la latencia del Middleware konker con 1 parámetro y 1000 usuarios

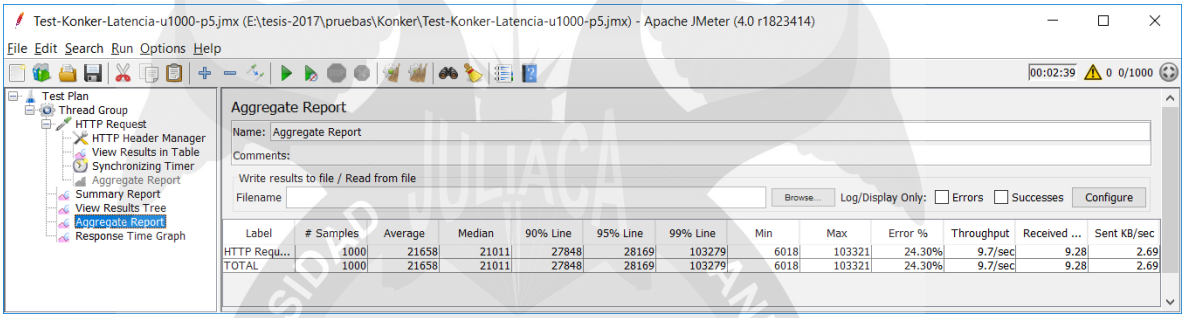


Fuente: Elaboración propia

4.9.17 Middleware konker con 5 parámetros y 1000 Usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 21011 ms.

Figura 69 Resultados de la latencia del Middleware konker con 5 parámetros y 1000 usuarios

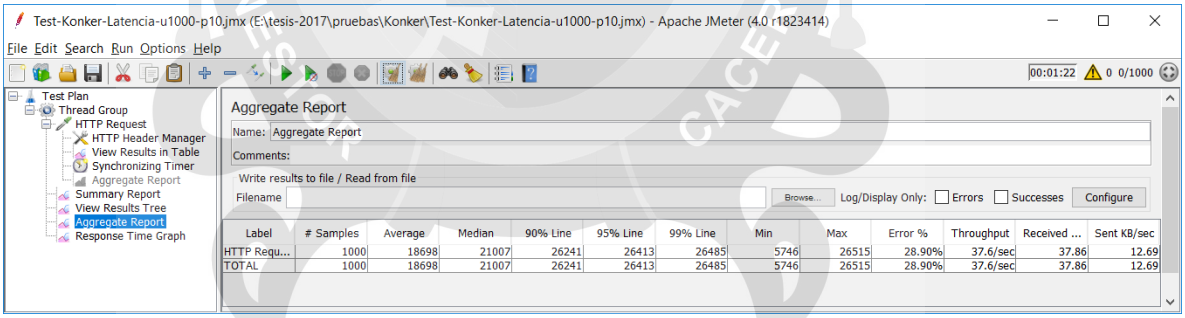


Fuente: Elaboración propia

4.9.18 Middleware konker con 10 parámetros y 1000 Usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 21007 ms.

Figura 70 Resultados de la latencia del Middleware konker con 10 parámetros y 1000 usuarios

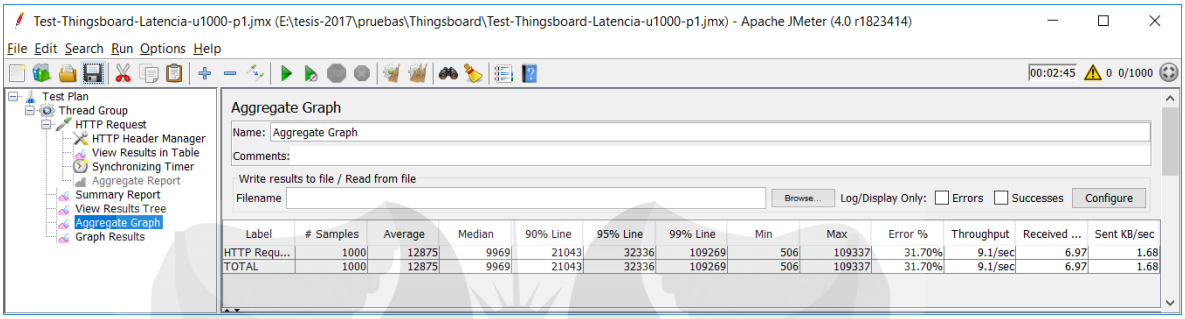


Fuente: Elaboración propia

4.9.19 Thingsboard con 1 parámetro y 1000 usuarios

En la figura, se toma la Columna de la Mediana de la Latencia, que es de 9969 ms.

Figura 71 Resultados de la latencia del Middleware ThingsBoard con 1 parámetro y 1000 usuarios

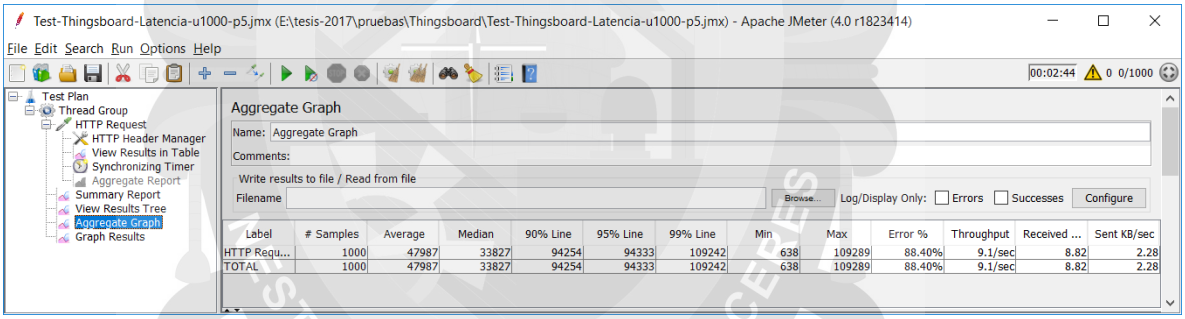


Fuente: Elaboración propia

4.9.20 Middleware thingsboard con 5 parámetros y 1000 usuarios

De la figura, se toma la Columna de la Mediana de la Latencia, que es de 33827 ms.

Figura 72 Resultados de la latencia del Middleware ThingsBoard con 5 parámetros y 1000 usuarios

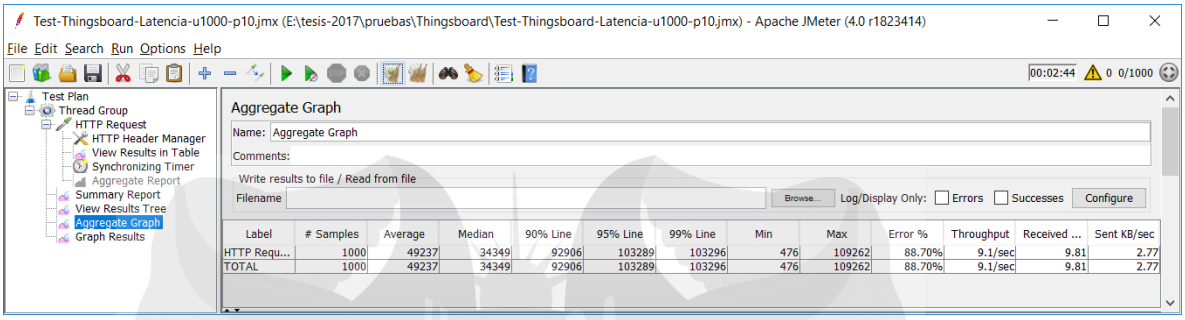


Fuente: Elaboración propia

4.9.21 Middleware thingsboard con 10 parámetros y 1000 usuarios

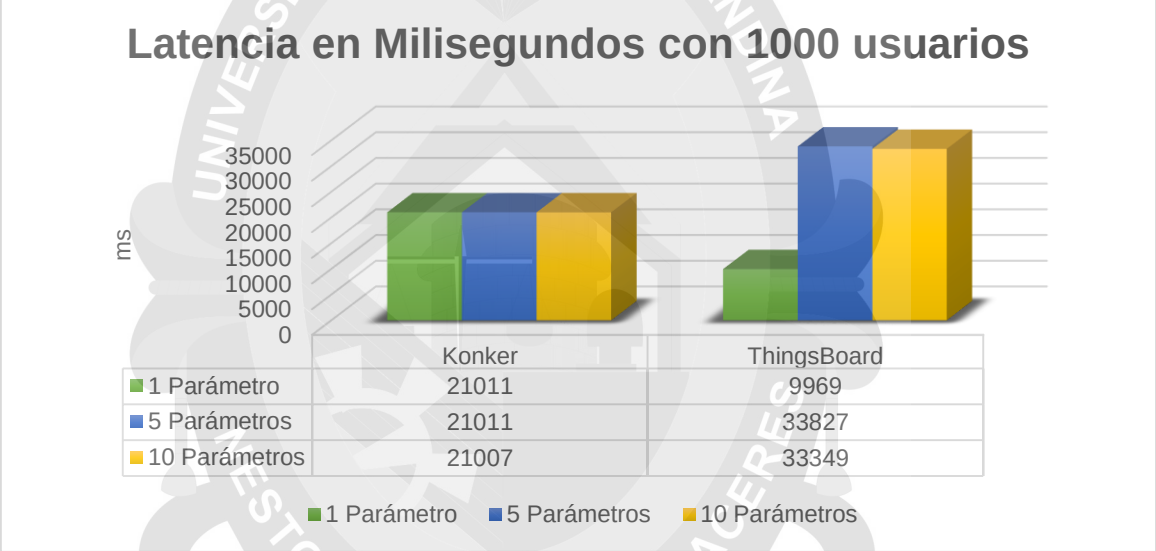
De la figura, se toma la Columna de la Mediana de la Latencia, que es de 34349 ms.

Figura 73 Resultados de la latencia del Middleware ThingsBoard con 10 parámetros y 1000 usuarios



Fuente: Elaboración propia

Figura 74 Gráfico de latencia con 1000 Usuarios



Fuente: Elaboración propia

En un escenario con 1000 usuarios concurrentes donde se enviaron 1, 5 y 10 parámetros, la plataforma Konker es la que tiene el menor tiempo a diferencia de ThingsBoard.

CONCLUSIONES

Primera. Los middlewares IoT Open-source alojados en el repositorio de Github están desarrollados en su mayoría en el lenguaje Java, son una buena alternativa en comparación de aquellos que se requiere realizar pagos por su uso.

Segunda. El índice de Medición del Servicio (SMI) acompañado de sus siete categorías, y sub categorías, se adapta para clasificar las métricas cuantitativas y cualitativas de un middleware IoT.

Tercera. Las métricas cuantitativas presentadas en este trabajo están basadas en bytes enviados y bytes recibidos, tiempo de conexión, porcentaje de error y latencia, consideramos que son características que se pueden medir según las pruebas, estas métricas son importantes para poder elegir la plataforma.

Cuarta. Concluimos que las métricas cualitativas son variables y están sujetas a una percepción personal quien va utilizar la plataforma middleware, están pueden tener muchos criterios y en realidad no son muy importantes en la elección de la plataforma, por lo tanto, la métrica cualitativa no representa significancia.

Quinta. Las métricas relacionadas con los bytes enviados y bytes recibidos no deben subestimarse, ya que es una herramienta valiosa para dimensionar la solución. El middleware ThingsBoard, es que tiene la menor cantidad de bytes enviados y recibidos en comparación de konker y podemos concluir que el número de parámetros no incrementa significativamente los bytes enviados.

Sexta. Concluimos que la métrica de tiempo de conexión no se incrementa el valor de milisegundos al aumentar el número de parámetros.

Séptima. Concluimos que el número de usuarios simultáneos que acceden al middleware es de 500, como valor óptimo, teniendo que el middleware konker no presenta errores en comparación de Thingsboard que presenta un 18%, siendo un valor aceptable.

Octava. Concluimos que la latencia tiene un comportamiento menor hasta los 500 usuarios, por lo tanto, es el número de usuarios que recomendamos que puedan enviar información simultáneamente a Thingsboard.

Novena. Concluimos que apache jmeter se desempeña muy bien como software de pruebas, para poder evaluar plataformas middleware IoT.

RECOMENDACIONES

Primera. Recomendamos seguir investigando en otras áreas de Internet de las cosas, como pueden ser, ciudades inteligentes, cuidado de la salud, agricultura de precisión, entre otros.

Segunda. Se recomienda evaluar los otros middlewares IoT open-source y realizar las pruebas de métricas cuantitativas como latencia medida en milisegundos, bytes enviados, bytes recibidos, tiempo de conexión medido en milisegundos, error de perdida de paquetes medida en porcentaje.

Tercera. Utilizar otras herramientas de prueba diferentes de Apache JMeter, o quizás desarrollar programas alternativos.

Cuarta. Recomendamos evaluar el tema de seguridad del middleware IoT, evaluando aspectos de autentificación de dispositivos, desde el punto de vista

REFERENCIAS BIBLIOGRÁFICAS

[1] J. Cardoso, C. Pereira, A. Aguiar, and R. Morla, "Benchmarking IoT middleware platforms," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2017, pp. 1–7.

[2] M. A. Amaro da Cruz, "Performance Evaluation of IoT Middleware," Instituto Nacional de Telecomunicaciones - INATEL, 2017.

[3] V. E. Araujo Soto, "Performance evaluation of scalable and distributed IoT platforms for smart regions," 2017.

[4] A. Salami and A. Yari, "A framework for comparing quantitative and qualitative criteria of IoT platforms," in *2018 4th International Conference on Web Research (ICWR)*, 2018, pp. 34–39.

[5] X. Larrucea, A. Combelles, J. Favaro, and K. Taneja, "Software Engineering for the Internet of Things," *IEEE Softw.*, vol. 34, no. 1, pp. 24–28, Jan. 2017.

[6] K. Ashton, "That 'Internet of Things' Thing," *RFID J.*, 2009.

[7] ITU, "Visión general de la Internet de las cosas (ITU-T Y.4000/Y.2060 (06/2012))," 2012.

[8] PcWorld, "Internet de las cosas: la nueva revolución industrial," 2015. [Online]. Available: http://www.pcworldenespanol.com/2015/03/06/internet-de-las-cosas-la-nueva-revolucion-industrial/. [Accessed: 17-Oct-2018].

[9] "Manual de Referencia de Lua 5.1." [Online]. Available: https://www.lua.org/manual/5.1/es/manual.html. [Accessed: 27-Sep-2018].

[10] Espressif Systems, "ESP8266EX Datasheet v5.9," 2018. [Online]. Available: https://www.espressif.com/en/subscribe. [Accessed: 27-Sep-2018].

[11] A. Branco, "Microcontroladores e Sistemas Embarcados," 2017.

[12] L. del Valle Hernández, "NodeMCU y el IoT tutorial paso a paso desde cero." [Online]. Available: https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/. [Accessed: 30-Sep-2018].

[13] adafruit learning system, "DHT11, DHT22 and AM2302 Sensors," USA, 2018.

[14] "How to Set Up the DHT11 Humidity Sensor on an Arduino." [Online]. Available: http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/. [Accessed: 30-Sep-2018].

[15] M. A. Abellán, "Software Arduino IDE - Curso de primeros pasos con Arduino." [Online]. Available: https://www.programoergosum.com/cursos-online/arduino/253-curso-de-iniciacion-a-arduino/software-arduino-ide. [Accessed: 17-Oct-2018].

[16] "Aprendiendo Arduino-IDE." [Online]. Available: https://aprendiendoarduino.wordpress.com/category/ide/. [Accessed: 30-Sep-2018].

[17] "Spanish – Open Source Hardware Association." [Online]. Available: https://www.oshwa.org/definition/spanish/. [Accessed: 30-Sep-2018].

[18] "The Open Source Definition (Annotated) | Open Source Initiative." [Online]. Available: https://opensource.org/osd-annotated. [Accessed: 28-Sep-2018].

[19] V. Kardeby, S. Forsström, P. Österberg, and U. Jennehag, "Fully Distributed Ubiquitous Information Sharing on a Global Scale for the Internet-of-Things," 2014.

[20] J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt, "Comparison of IoT platform architectures: A field study based on a reference architecture," in *2016 Cloudification of the Internet of Things (CIoT)*, 2016, pp. 1–6.

[21] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel, "Survey of platforms for massive IoT," in *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, 2018, pp. 1–8.

[22] mqtt.org, "MQTT." [Online]. Available: http://mqtt.org/. [Accessed: 17-Oct-2018].

[23] "chorevolution.eu: A Solution for Distributed IoT-Enabled Applications (Follow.A_Solution_for_Distributed_IoT_Enabled_Applications_Article)." [Online]. Available: http://www.chorevolution.eu/bin/view/Follow/A_Solution_for_Distributed_IoT_Enabled_Applications_Article. [Accessed: 13-Jul-2018].

[24] "DeviceHive - Open Source IoT Data Platform with the wide range of integration options." [Online]. Available: https://devicehive.com/. [Accessed: 14-Jul-2018].

[25] "Open Source IoT Platform & Toolkit | DSA - Home." [Online]. Available: http://iot-dsa.org/. [Accessed: 14-Jul-2018].

[26] "Home - IoTivity." [Online]. Available: https://iotivity.org/. [Accessed: 13-Jul-2018].

[27] Fiware, "About Us - FIWARE." [Online]. Available: https://www.fiware.org/about-us/. [Accessed: 05-Jul-2018].

[28] Fiware, "Internet of Things (IoT) Services Enablement Architecture." [Online]. Available: http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Internet_of_Things_%28IoT%29_Services_Enablement_Architecture. [Accessed: 05-Jul-2018].

[29] "Platform - IoT-Ignite." [Online]. Available: https://www.iot-ignite.com/platform/. [Accessed: 13-Jul-2018].

[30] "Technical Architecture - IoT-Ignite Devzone." [Online]. Available: https://devzone.iot-ignite.com/knowledge-base/iot-ignite-technical-architecture/. [Accessed: 13-Jul-2018].

[31] "Kaa open-source IoT platform." [Online]. Available: https://www.kaaproject.org/. [Accessed: 13-Jul-2018].

[32] "Architecture overview · Kaa." [Online]. Available: https://kaaproject.github.io/kaa/docs/v0.10.0/Architecture-overview/. [Accessed: 13-Jul-2018].

[33] "Eclipse Kapua." [Online]. Available: https://www.eclipse.org/kapua/. [Accessed: 14-Jul-2018].

[34] "Kapua source code." [Online]. Available: https://github.com/eclipse/kapua. [Accessed: 14-Jul-2018].

[35] "Eclipse Kapua – Architecture." [Online]. Available: https://www.eclipse.org/community/eclipse_newsletter/2017/march/article4.php. [Accessed: 14-Jul-2018].

[36] "Konker - A sua solução de IoT de forma ágil e sem complicação." .

[37] "LinkSmart® - Free, Open Source IoT Platform." .

[38] "Architecture - LinkSmart® Device Gateway - LinkSmart® Docs." [Online]. Available: https://docs.linksmart.eu/display/DGW/Architecture. [Accessed: 13-Jul-2018].

[39] "OpenIoT." [Online]. Available: https://github.com/OpenIotOrg/openiot. [Accessed: 13-Jul-2018].

[40] "OpenIoT Architecture," 2013. [Online]. Available: https://github.com/OpenIotOrg/openiot/wiki/OpenIoT-Architecture. [Accessed: 13-Jul-2018].

[41] OpenMTC, "OpenMTC Platform." [Online]. Available: http://www.openmtc.org/platform.html. [Accessed: 05-Jul-2018].

[42] "OpenRemote | Open Source for Internet of Things." [Online]. Available: http://www.openremote.com/. [Accessed: 14-Jul-2018].

[43] "Openremote source code." [Online]. Available: https://github.com/openremote/. [Accessed: 14-Jul-2018].

[44] SiteWhere, "SiteWhere Documentation System Overview." .

[45] SiteWhere, "SiteWhere System Architecture." [Online]. Available: http://documentation.sitewhere.io/architecture.html. [Accessed: 05-Jul-2018].

[46] Indra, "Origen y Evolución — Sofia2 1.0 documentation," 2018. [Online]. Available: https://sofia2.readthedocs.io/en/latest/origenyevolucion.html. [Accessed: 31-Oct-2018].

[47] Indra, "Sofia2 Welcome Pack." [Online]. Available: http://sofia2.com/docs/Sofia2-Welcome Pack-v03.pdf. [Accessed: 05-Jul-2018].

[48] INDRA, "SOFIA2 : Capacidades." [Online]. Available: http://sofia2.com/home.html. [Accessed: 05-Jul-2018].

[49] Indra, "Arquitectura global — Sofia2 1.0 documentation," 2018. [Online]. Available: https://sofia2.readthedocs.io/en/latest/arquitectura_global.html#. [Accessed: 31-Oct-2018].

[50] "The Things Network." [Online]. Available: https://www.thethingsnetwork.org/.

[Accessed: 14-Jul-2018].

[51] "The Things Networks source code." [Online]. Available: https://github.com/thethingsnetwork. [Accessed: 14-Jul-2018].

[52] "Thinger.io – Open Source IoT Platform." [Online]. Available: https://thinger.io/. [Accessed: 14-Jul-2018].

[53] "Thinger – Source Coude." [Online]. Available: https://github.com/thinger-io. [Accessed: 14-Jul-2018].

[54] "ThingsBoard - Open-source IoT Platform." [Online]. Available: https://thingsboard.io/. [Accessed: 13-Jul-2018].

[55] "ThingsBoard Architecture." [Online]. Available: https://thingsboard.io/docs/reference/architecture/#clustering-mode. [Accessed: 13-Jul-2018].

[56] J. Guth *et al.*, "A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences," Springer, Singapore, 2018, pp. 81–101.

[57] "WSO2 -The Open Source Technology for Digital Business." .

[58] "Architecture - Identity Server 5.3.0 - WSO2 Documentation." [Online]. Available: https://docs.wso2.com/display/IS530/Architecture. [Accessed: 13-Jul-2018].

[59] C. Swaney, "Carnegie Mellon Silicon Valley Researchers Collaborate with Industry to Develop Measures of Quality and Performance for Cloud-Computing Services," 2010. [Online]. Available: https://web.archive.org/web/20101105122335/http://www.cit.cmu.edu/media/press/2010/05_16_sv_cloud_computing.html. [Accessed: 15-Oct-2018].

[60] CSMIC, "Service Measurement Index Framework Version 2.1," 2014. [Online]. Available: http://csmic.org/downloads/SMI_Overview_TwoPointOne.pdf. [Accessed: 16-Oct-2018].

[61] M. G. Piattini Velthuis, F. O. García Rubio, and I. Caballero Muñoz-Reja, *Calidad de sistemas informáticos*. Alfaomega, 2007.

[62] "ISO/IEC 25000:2014(en), Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE." [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-

iec:25000:ed-2:v1:en. [Accessed: 16-Oct-2018].

[63] J. C. Moreno *et al.*, *Evaluación Temprana de la Usabilidad Empleando Patrones Embebidos en la Construcción del Modelo Conceptual para Aplicaciones Web*. .

[64] C. Bormann, K. Hartke, and Z. Shelby, "The Constrained Application Protocol (CoAP)," *rfc7252*, 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7252.txt. [Accessed: 16-Jul-2018].

[65] R. K. Ghosh, *Wireless Networking and Mobile Data Management*. Singapore: Springer Singapore, 2017.

[66] R. Hernández Sampieri, C. Fernández Collado, and P. Baptista Lucio, *Metodología de la investigación*, 6ta ed. McGraw-Hill Education, 2014.

[67] XMeter, "MQTT JMeter Plugin," 2018. [Online]. Available: https://github.com/emqx/mqtt-jmeter. [Accessed: 16-Oct-2018].

[68] M. Noguchi, "A Gatling stress test plugin for MQTT," 2016. [Online]. Available: https://github.com/mnogu/gatling-mqtt. [Accessed: 16-Oct-2018].

[69] "Apache JMeter - Apache JMeter™." [Online]. Available: http://jmeter.apache.org/. [Accessed: 16-Jul-2018].

[70] Lawrence Miller, *Public PaaS for Dummies: Second Edition Guide*. Oracle, 2017.

[71] "¿Quiénes somos? - OVH." [Online]. Available: https://www.ovh.com/world/es/quienes-somos/. [Accessed: 17-Oct-2018].

APÉNDICE

Instalación del Middleware ThingsBoard

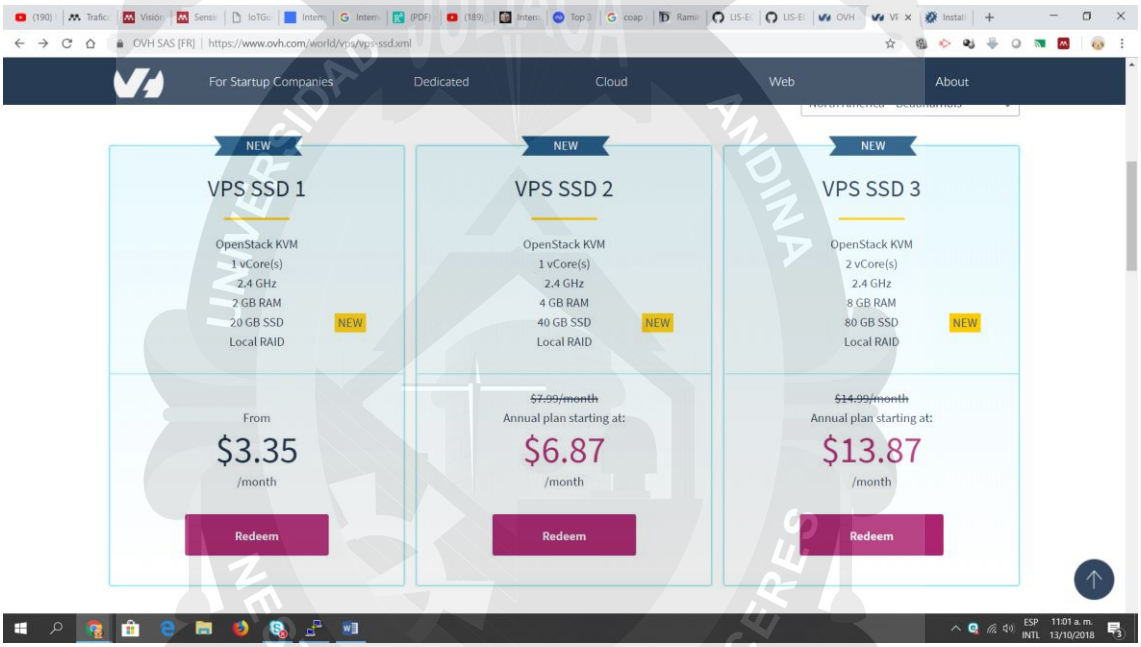
1. Contratar VPS

   Contratar el VPS, elegir el VPS SSD2, de la empresa OVH.

   https://www.ovh.com/world/es/

   Tiene las siguiente características

   4 Gb de RAM, 40 Gb SSD Costo Mensual de US$ 6.87



2. Elegir el Sistema Operativo, para nuestro caso debe ser Ubuntu 16.04.

3. Instalar el Java en el prompt escribir cada una de las siguientes instrucciones:

   $ apt-get -f install

   $ sudo apt-get install default-jre

   $ sudo apt-get install default-jdk

   $ sudo add-apt-repository ppa:webupd8team/java

   $ sudo apt-get update

   $ sudo apt-get install oracle-java8-installer

4. En el terminal, escribir el siguiente commando:

```
wget
https://github.com/thingsboard/thingsboard/releases/download/
v2.1/thingsboard-2.1.deb
```

5. A continuación el siguiente comando

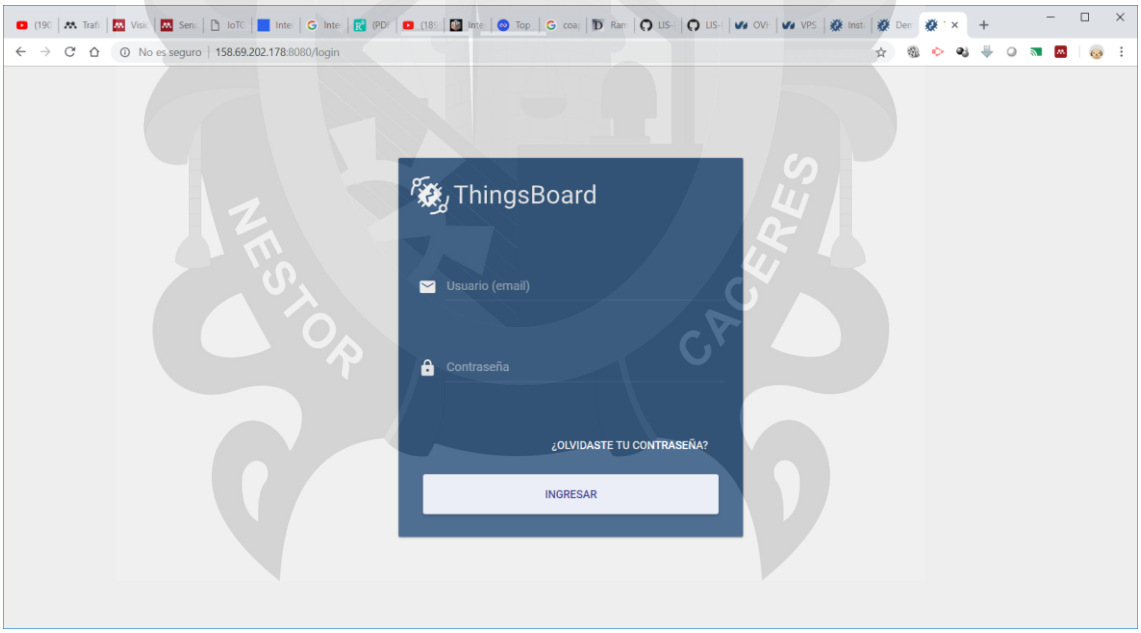
```
sudo dpkg -i thingsboard-2.1.deb
```

6. Iniciar el thingsboard con el siguiente comando:

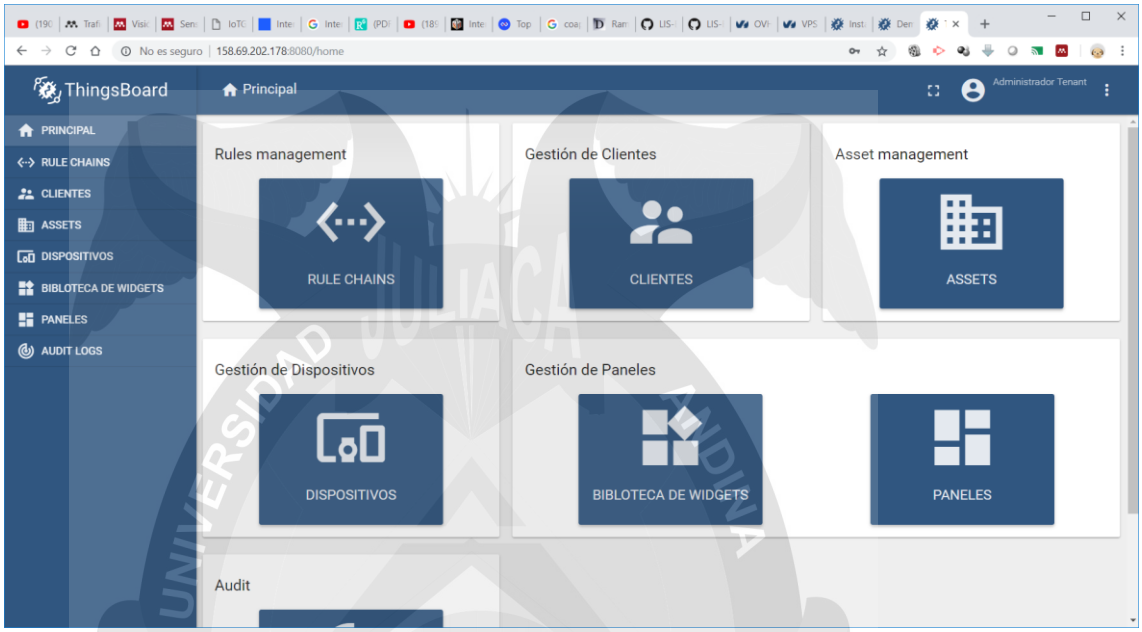
```
sudo service thingsboard start
```

7. En el navegador deberás de escribir la dirección IP del VPS
   http://158.69.202.177:8080/

   Login: **tenant@thingsboard.org**
   Password: **tenant**



8. Al colocar debe aparece la ventana principal con las diferentes opciones, del middleware ThingsBoard.



Los archivos de las pruebas se encuentran en el repositorio github en el siguiente url:

https://github.com/arroyopaz